OpenShift Container Platform 4.16

セキュリティーおよびコンプライアンス

OpenShift Container Platform のセキュリティーに関する理解および管理

Last Updated: 2025-12-06

OpenShift Container Platform 4.16 セキュリティーおよびコンプライアンス

OpenShift Container Platform のセキュリティーに関する理解および管理

Abstract

このドキュメントでは、クラスターのセキュリティー保護に役立つコンテナーのセキュリティー、証明書の設定、および暗号化の有効化を説明します。

Table of Contents

第1章 OPENSHIFT CONTAINER PLATFORM のセキュリティーおよびコンプライアンス

1.1. セキュリティーの概要

OpenShift Container Platform クラスターの各種の側面を適切に保護する方法を理解しておくことが重要です。

1.1.1. コンテナーのセキュリティー

OpenShift Container Platform セキュリティーを理解するためのスタート地点として、コンテナーのセキュリティーについて の概念を確認すると良いでしょう。このセクションとこの後のセクションでは、OpenShift Container Platform で有効なコンテナーのセキュリティー対策に関する概要を説明します。これには、ホスト層、コンテナーとオーケストレーション層、およびビルドとアプリケーション層の各種ソリューションが含まれます。これらのセクションでは、以下のトピックについても説明します。

- コンテナーのセキュリティーが重要である理由、および既存のセキュリティー標準との違い。
- ホスト (RHCOS および RHEL) 層で提供されるコンテナーのセキュリティー対策と OpenShift Container Platform で提供されるコンテナーのセキュリティー対策。
- 脆弱性についてコンテナーのコンテンツとソースを評価する方法。
- コンテナーのコンテンツをプロアクティブに検査できるようにビルドおよびデプロイメントプロセスを設計する方法。
- 認証および認可によってコンテナーへのアクセスを制御する方法。
- OpenShift Container Platform でネットワークと割り当て済みストレージのセキュリティーを保護する方法。
- API 管理および SSO のコンテナー化ソリューション。

1.1.2. 監査

OpenShift Container Platform 監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。管理者は 監査ログポリシーの設定 と 監査ログの表示 が可能です。

1.1.3. 証明書

証明書は、クラスターへのアクセスを検証するためにさまざまなコンポーネントによって使用されます。管理者は、デフォルトの Ingress 証明書の置き換え、API サーバー証明書の追加、または サービス証明書の追加 が可能です。

クラスターで使用される証明書の種類の詳細を確認することもできます。

- API サーバーのユーザーによって提供される証明書
- プロキシー証明書
- サービス CA 証明書
- ノード証明書

- ブートストラップ証明書
- etcd 証明書
- OLM 証明書
- 集合 API クライアント証明書
- Machine Config Operator 証明書
- デフォルト ingress のユーザーによって提供される証明書
- Ingress 証明書
- モニタリングおよびクラスターロギング Operator コンポーネント証明書
- コントロールプレーンの証明書

1.1.4. データの暗号化

クラスターの etcd 暗号化を有効 にして、データセキュリティーのレイヤーを追加で提供することができます。たとえば、etcd バックアップが不正な関係者に公開された場合に機密データの損失を防ぐのに役立ちます。

1.1.5. 脆弱性スキャン

管理者は Red Hat Quay Container Security Operator を使用して vulnerability scans を実行し、検出された脆弱性の情報を確認できます。

1.2. コンプライアンスの概要

多くの OpenShift Container Platform のお客様においては、システムが実稼働環境で使用される前に、一定レベルでの規制への対応またはコンプライアンスが必要になります。この規制対応は、国家標準、業界標準または組織の企業ガバナンスフレームワークによって課せられます。

1.2.1. コンプライアンスの確認

管理者は Compliance Operator を使用してコンプライアンススキャンを実行し、検出された問題の修復を提案できます。**oc-compliance** プラグイン は、Compliance Operator を簡単に操作するための一連のユーティリティーを提供する OpenShift CLI (**oc**) プラグインです。

1.2.2. ファイルの整合性チェック

管理者は File Integrity Operator を使用して、クラスターノードでファイルの整合性チェックを継続的に実行し、変更されたファイルのログを提供できます。

1.3. 関連情報

- 認証について
- 内部 OAuth サーバーの設定
- アイデンティティープロバイダー設定について

- RBAC の使用によるパーミッションの定義および適用
- Security Context Constraints の管理

第2章 コンテナーのセキュリティー

2.1. コンテナーのセキュリティーについて

コンテナー化されたアプリケーションのセキュリティー保護においては、複数のセキュリティーレベルが関係します。

- コンテナーのセキュリティーは、信頼できるベースコンテナーイメージから始まり、CI/CD パイプラインを通過するためにコンテナーのビルドプロセスまで適用されます。

  重要

  デフォルトでは、イメージストリームは自動的に更新されません。このデフォルトの動作では、イメージストリームによって参照されるイメージに対するセキュリティー更新は自動的に行われないため、セキュリティーの問題が発生する可能性があります。このデフォルト動作をオーバーライド方法の詳細は、イメージストリームタグの定期インポートの設定 を参照してください。

- コンテナーがデプロイされると、そのセキュリティーはセキュアなオペレーティングシステムやネットワーク上で実行されているかどうかに依存し、コンテナー自体とこれと対話するユーザーやホスト間に明確な境界を確立することが必要です。

- セキュリティーを継続して保護できるかどうかは、コンテナーイメージをスキャンして脆弱性の有無を確認でき、脆弱なイメージを効率的に修正し、置き換える効率的な方法があるかどうかに依存します。

OpenShift Container Platform などのプラットフォームが追加設定なしで提供する内容のほかに、各組織には独自のセキュリティー需要がある可能性があります。OpenShift Container Platform をデータセンターにデプロイする前にも、一定レベルのコンプライアンス検証が必要になる場合があります。

同様に、独自のエージェント、特殊ハードウェアドライバーまたは暗号化機能を OpenShift Container Platform に追加して組織のセキュリティー基準を満たす必要がある場合があります。

このガイドでは、OpenShift Container Platform で有効なコンテナーのセキュリティー対策に関する概要を説明します。これには、ホスト層、コンテナーとオーケストレーション層、およびビルドとアプリケーション層の各種ソリューションが含まれます。次に、これらのセキュリティー対策を実行するのに役立つ特定の OpenShift Container Platform ドキュメントを参照します。

このガイドには、以下の情報が記載されています。

- コンテナーのセキュリティーが重要である理由、および既存のセキュリティー標準との違い。
- ホスト (RHCOS および RHEL) 層で提供されるコンテナーのセキュリティー対策と OpenShift Container Platform で提供されるコンテナーのセキュリティー対策。
- 脆弱性についてコンテナーのコンテンツとソースを評価する方法。
- コンテナーのコンテンツをプロアクティブに検査できるようにビルドおよびデプロイメントプロセスを設計する方法。
- 認証および認可によってコンテナーへのアクセスを制御する方法。
- OpenShift Container Platform でネットワークと割り当て済みストレージのセキュリティーを保護する方法。

- API 管理および SSO のコンテナー化ソリューション。

このガイドの目的は、コンテナー化されたワークロードに OpenShift Container Platform を使用するセキュリティー上の重要な利点と、Red Hat エコシステム全体がコンテナーのセキュリティーを確保し、維持する際にどのようなロールを果たしているかについて理解を促すことにあります。また、OpenShift Container Platform の使用により組織のセキュリティー関連の目標を達成する方法を理解するのに役立ちます。

2.1.1. コンテナーについて

コンテナーは、アプリケーションとそのすべての依存関係を 1 つのイメージにパッケージ化します。このイメージは、変更なしに開発環境からテスト環境、実稼働環境へとプロモートすることができます。コンテナーは、他のコンテナーと密接に動作する大規模なアプリケーションの一部である可能性があります。

コンテナーは、複数の環境、および物理サーバー、仮想マシン (VM)、およびプライベートまたはパブリッククラウドなどの複数のデプロイメントターゲット間に一貫性をもたらします。

コンテナーを使用するメリットには以下が含まれます。

インフラストラクチャー	アプリケーション
共有される Linux オペレーティングシステムのカーネル上でのアプリケーションプロセスのサンドボックス化	アプリケーションとそのすべての依存関係のパッケージ化
仮想マシンを上回る単純化、軽量化、高密度化の実現	すべての環境に数秒でデプロイが可能。CI/CD の実現
複数の異なる環境間での移植性	コンテナー化されたコンポーネントへのアクセスと共有が容易になる

Linux コンテナーの詳細は、Red Hat カスタマーポータル上にある Understanding Linux containers を参照してください。RHEL コンテナーの詳細は、RHEL 製品ドキュメントの Building, running, and managing containers を参照してください。

2.1.2. OpenShift Container Platform について

コンテナー化されたアプリケーションがデプロイされ、実行され、管理される方法を自動化することは、OpenShift Container Platform をはじめとするプラットフォームのジョブです。OpenShift Container Platform は、コアとして Kubernetes プロジェクトに依存し、スケーラブルなデータセンターの多数のノード間でコンテナーをオーケストレーションするエンジンを提供します。

Kubernetes は、複数の異なるオペレーティングシステムおよびアドオンのコンポーネントを使用して実行できるプロジェクトです。これらのオペレーティングシステムおよびアドオンコンポーネントは、プロジェクトでのサポート容易性を保証していません。そのため、Kubernetes プラットフォームによって、セキュリティーの内容が異なる可能性があります。

OpenShift Container Platform は、Kubernetes セキュリティーをロックダウンし、プラットフォームを各種の拡張コンポーネントと統合するように設計されています。このため、OpenShift Container Platform は、オペレーティングシステム、認証、ストレージ、ネットワーク、開発ツール、ベースコンテナーイメージ、その他の多くのコンポーネントを含む、各種オープンソース技術の大規模な Red Hat エコシステムを利用します。

OpenShift Container Platform には、プラットフォーム自体およびプラットフォーム上で実行されるコンテナー化されたアプリケーションの脆弱性の発見、およびその脆弱性に対する修正の迅速なデプロイにおける Red Hat の豊富な経験が最大限に活用されます。また、Red Hat は、新規コンポーネントが利用可能になる時点でそれらのコンポーネントを OpenShift Container Platform に効率的に統合し、各種テクノロジーをお客様の個々のニーズに適応させる点においても多くの経験があります。

関連情報

- OpenShift Container Platform アーキテクチャー
- OpenShift セキュリティーガイド

2.2. ホストおよび仮想マシンのセキュリティーについて

コンテナーと仮想マシンはいずれも、ホストで実行されているアプリケーションをオペレーティングシステム自体から分離する方法を提供します。RHCOS (OpenShift Container Platform で使用されるオペレーティングシステム) に関する理解は、ホストシステムがコンテナーおよびホストを相互から保護する方法を確認する際に役立ちます。

2.2.1. Red Hat Enterprise Linux CoreOS (RHCOS) でのコンテナーのセキュリティー保護

コンテナーは、それぞれのコンテナーを起動するために同じカーネルおよびコンテナーランタイムを使用して、同じホストで実行される多数のアプリケーションのデプロイメントを単純化します。アプリケーションは多くのユーザーが所有できます。これらのアプリケーションを分離した状態に維持し、これらのアプリケーションの別々のバージョン、また互換性のないバージョンも問題なく同時に実行できるためです。

Linux では、コンテナーは特殊なタイプのプロセスにすぎないため、コンテナーのセキュリティーを保護することは、他の実行中のプロセスを保護することと多くの点で似ています。コンテナーを実行する環境は、オペレーティングシステムで起動します。このオペレーティングシステムでは、ホストで実行しているコンテナーや他のプロセスからホストカーネルのセキュリティーを保護するだけでなく、複数のコンテナーのセキュリティーを相互から保護できる必要があります。

OpenShift Container Platform 4.16 は RHCOS ホストで実行され、Red Hat Enterprise Linux (RHEL) をワーカーノードとして使用するオプションが指定されるため、デフォルトで以下の概念がデプロイされた OpenShift Container Platform クラスターに適用されます。これらの RHEL セキュリティー機能は、OpenShift Container Platform で実行中のコンテナーのセキュリティーを強化するためのコアとなる機能です。

- **Linux 名前空間** は、特定のグローバルなシステムリソースの抽象化を作成し、名前空間内のプロセスに対して、そのリソースが独立したインスタンスであるかのように見せることを可能にします。これにより、複数のコンテナーが競合せずに同じコンピューティングリソースを同時に使用することができます。デフォルトでホストから分離されているコンテナー名前空間として、マウントテーブル、プロセステーブル、ネットワークインターフェイス、ユーザー、コントロールグループ、UTS、および IPC 名前空間があります。ホストの名前空間への直接アクセスが必要なコンテナーには、そのアクセスを要求するための昇格された権限が必要です。名前空間の種類の詳細は、RHEL 9 コンテナーのドキュメントの コンテナーの構築、実行、および管理 を参照してください。
- **SELinux** はセキュリティーの層を追加し、コンテナーを相互に、またホストから分離させます。SELinux により、管理者は、それぞれのユーザー、アプリケーション、プロセスおよびファイルに対して強制アクセス制御 (MAC) を実施できます。

> **警告**
>
> RHCOS での SELinux の無効化はサポートされていません。

- **CGroup** (コントロールグループ) はプロセスのコレクションに関するリソースの使用 (CPU、メモリー、ディスク I/O、ネットワークなど) を制限し、設定し、分離します。CGroup は、同じホスト上のコンテナーが相互に影響を与えないようにするために使用されます。
- **Secure computing mode (seccomp)** プロファイルは、利用可能なシステム呼び出しを制限するためにコンテナーに関連付けることができます。seccomp の詳細は、Red Hat OpenShift security guide の 94 ページを参照してください。
- **RHCOS** を使用したコンテナーのデプロイは、ホスト環境を最小化してコンテナー向けに調整することで、攻撃される対象の規模を縮小します。CRI-O コンテナーエンジン は、デスクトップ指向のスタンドアロン機能を実装する他のコンテナーエンジンとは対照的に、Kubernetes および OpenShift Container Platform が必要とする機能のみを実装してコンテナーを実行し、管理することで、その攻撃対象領域をさらに削減します。

RHCOS は、OpenShift Container Platform クラスターでコントロールプレーン (マスター) およびワーカーノードとして機能するように特別に設定された Red Hat Enterprise Linux (RHEL) のバージョンです。そのため、RHCOS は、Kubernetes および OpenShift Container Platform サービスと共にコンテナーのワークロードを効率的に実行するように調整されます。

> **注記**
>
> OpenShift Container Platform クラスターの RHCOS システムをさらに保護するには、ホストシステム自体の管理またはモニタリングを行うコンテナーを除き、ほとんどのコンテナーを root 以外のユーザーとして実行する必要があります。権限レベルを下げたり、付与する権限を可能な限り低くしてコンテナーを作成することが、独自の OpenShift Container Platform クラスターを保護する方法として推奨されます。

関連情報

- ノードによるリソースの制約の適用方法
- Security Context Constraints の管理
- OpenShift クラスターでサポートされるプラットフォーム
- ユーザーによってプロビジョニングされるインフラストラクチャーを使用したクラスターの要件
- RHCOS の設定方法の選択
- Ignition
- カーネル引数
- カーネルモジュール
- ディスクの暗号化

- Chrony タイムサービス
- OpenShift Update Service について
- FIPS 暗号

2.2.2. 仮想化とコンテナーの比較

従来の仮想化は、アプリケーション環境を同じ物理ホスト上で分離させた状態にするためのもう 1 つの方法です。ただし、仮想マシンはコンテナーとは異なる方法で動作します。仮想化は、ゲスト仮想マシン (VM) を起動するハイパーバイザーを使用します。仮想マシンにはそれぞれ、実行中のカーネルで代表される独自のオペレーティングシステム (OS) のほか、実行されるアプリケーションとその依存関係があります。

仮想マシンの場合、ハイパーバイザーはゲスト同士を分離させ、ゲストをホストカーネルから分離します。ハイパーバイザーにアクセスする個々のユーザーおよびプロセスの数は少ないため、物理サーバーで攻撃される対象の規模が縮小します。ただし、この場合もセキュリティーの監視が依然として必要になります。あるゲスト仮想マシンがハイパーバイザーのバグを利用して、別の仮想マシンまたはホストカーネルにアクセスできる可能性があります。また、OS にパッチを当てる必要がある場合は、その OS を使用するすべてのゲスト仮想マシンにパッチを当てる必要があります。

コンテナーはゲスト仮想マシン内で実行可能であり、これが必要になる場合のユースケースもあるでしょう。たとえば、リフトアンドシフト方式でアプリケーションをクラウドに移行するなど、コンテナーに従来型のアプリケーションをデプロイする場合などです。

しかし、単一ホストでのコンテナーの分離は、柔軟性があり、スケーリングしやすいデプロイメントソリューションを提供します。このデプロイメントモデルは、クラウドネイティブなアプリケーションにとくに適しています。コンテナーは通常、仮想マシンよりもはるかに小さいため、メモリーと CPU の消費量が少なくなります。

コンテナーと仮想マシンの違いは、RHEL 7 コンテナードキュメントの Linux Containers Compared to KVM Virtualization を参照してください。

2.2.3. OpenShift Container Platform のセキュリティー保護

OpenShift Container Platform をデプロイする際に、インストーラーでプロビジョニングされるインフラストラクチャー (利用可能ないくつかのプラットフォーム) またはユーザーによってプロビジョニングされるインフラストラクチャーを選択できます。FIPS モードの有効化や初回の起動時に必要なカーネルモジュールの追加など、低レベルのセキュリティー関連の設定は、ユーザーによってプロビジョニングされるインフラストラクチャーの場合に役立つ場合があります。同様に、ユーザーによってプロビジョニングされるインフラストラクチャーは、非接続の OpenShift Container Platform デプロイメントに適しています。

セキュリティーが強化され、OpenShift Container Platform に他の設定変更が行われる場合、以下を含む目標を明確にするようにしてください。

- 基礎となるノードを可能な限り汎用的な状態で維持する。同様のノードをすぐ、かつ指定した方法で破棄したり起動したりできるようにする必要があります。
- ノードに対して直接的に 1 回限りの変更を行うのではなく、OpenShift Container Platform でのノードへの変更をできる限り管理する。

上記を目標とすると、ほとんどのノードの変更はインストール時に Ignition で行うか、Machine Config Operator によってノードのセットに適用される MachineConfig を使用して後で行う必要があります。この方法で実行できるセキュリティー関連の設定変更の例を以下に示します。

- カーネル引数の追加
- カーネルモジュールの追加
- FIPS 暗号のサポートの有効化
- ディスク暗号化の設定
- chrony タイムサービスの設定

Machine Config Operator のほかにも、Cluster Version Operator (CVO) によって管理される OpenShift Container Platform インフラストラクチャーの設定に使用できる他の Operator が複数あります。CVO は、OpenShift Container Platform クラスター更新の多くの部分を自動化できます。

関連情報

- FIPS 暗号

2.3. RHCOS のハードニング

RHCOS は、OpenShift Container Platform にデプロイするように作成され、調整されました。RHCOS ノードへの変更はほとんど不要です。OpenShift Container Platform を採用するすべての組織には、システムハードニングに関する独自の要件があります。OpenShift 固有の変更および機能 (Ignition、ostree、読み取り専用 /**usr** など) が追加された RHEL システムとして、RHCOS を RHEL システムと同様に強化できます。ハードニングの管理方法には違いがあります。

OpenShift Container Platform およびその Kubernetes エンジンの主要機能は、必要に応じてアプリケーションおよびインフラストラクチャーを迅速にスケールアップおよびダウンできることです。避けられない状況でない限り、ホストにログインしてソフトウェアを追加したり設定を変更したりして RHCOS に直接変更を加える必要はありません。OpenShift Container Platform インストーラーおよびコントロールプレーンで RHCOS への変更を管理し、手動による介入なしに新規ノードを起動できるようにする必要があります。

そのため、独自のセキュリティー上のニーズに対応するために OpenShift Container Platform で RHCOS ノードをハードニングする場合、ハードニングする内容とハードニング方法の両方を考慮する必要があります。

2.3.1. RHCOS のハードニング対象の選択

RHEL システムのセキュリティーへの取り組み方については、RHEL 9 のセキュリティーの強化 ガイドを参照してください。

このガイドでは、暗号化のアプローチ、脆弱性の評価方法、および各種サービスへの脅威の評価方法を説明します。また、コンプライアンス基準に関するスキャン、ファイルの整合性の確認、監査の実行、およびストレージデバイスの暗号化の方法を確認することができます。

ハードニングする機能の理解に基づいて、RHCOS でそれらをハードニングする方法を決定することができます。

2.3.2. RHCOS のハードニング方法の選択

OpenShift Container Platform での RHCOS システムの直接的な変更は推奨されません。代わりに、ワーカーノードやコントロールプレーンノードなどのノードのプールにあるシステムを変更することを考慮する必要があります。新規ノードが必要な場合、ベアメタル以外のインストールでは、必要なタイ

プの新規ノードを要求でき、ノードは RHCOS イメージおよび先に行った変更に基づいて作成されます。

インストール前や、インストール時、およびクラスターの稼働後に RHCOS を変更することができます。

2.3.2.1. インストール前のハードニング

ベアメタルのインストールでは、OpenShift Container Platform のインストールを開始する前にハードニング機能を RHCOS に追加できます。たとえば、RHCOS インストーラーの起動時に、SELinux ブール型や対称マルチスレッドなどの各種の低レベル設定などのセキュリティー機能をオンまたはオフにするためにカーネルオプションを追加できます。

> **警告**
>
> RHCOS ノードでの SELinux の無効化はサポートされていません。

ベアメタル RHCOS のインストールの場合は難易度が上がりますが、この場合、OpenShift Container Platform インストールを開始する前にオペレーティングシステムの変更を取得することができます。これは、ディスクの暗号化や特別なネットワーク設定など、特定の機能を可能な限り早期に設定する必要がある場合に重要になります。

2.3.2.2. インストール時のハードニング

OpenShift Container Platform インストールプロセスを中断し、Ignition 設定を変更できます。Ignition 設定を使用して、独自のファイルおよび systemd サービスを RHCOS ノードに追加できます。また、インストールに使用する **install-config.yaml** ファイルに基本的なセキュリティー関連の変更を加えることもできます。この方法で追加した内容は、各ノードの初回起動時に利用できます。

2.3.2.3. クラスターの実行後のハードニング

OpenShift Container Platform クラスターの起動後にハードニング機能を RHCOS に適用する方法は複数あります。

- デーモンセット: すべてのノードでサービスを実行する必要がある場合は、そのサービスを Kubernetes **DaemonSet** オブジェクト で追加できます。
- マシン設定: **MachineConfig** オブジェクトには、同じ形式の Ignition 設定のサブセットが含まれます。マシン設定をすべてのワーカーノードまたはコントロールプレーンノードに適用することで、クラスターに追加される同じタイプの次のノードで同じ変更が適用されるようにできます。

ここで説明しているすべての機能は、OpenShift Container Platform の製品ドキュメントに記載されています。

関連情報

- OpenShift セキュリティーガイド
- RHCOS の設定方法の選択

- ノードの変更
- インストール設定ファイルの手動作成
- Kubernetes マニフェストおよび Ignition 設定ファイルの作成
- ISO イメージを使用した RHCOS のインストール
- ノードのカスタマイズ
- カーネル引数のノードへの追加
- オプションの設定パラメーター
- FIPS 暗号のサポート
- RHEL コア crypto コンポーネント

2.4. コンテナーイメージの署名

Red Hat は、Red Hat Container Registry でイメージの署名を提供します。これらの署名は、Machine Config Operator (MCO) を使用して OpenShift Container Platform 4 クラスターにプルされる際に自動的に検証されます。

Quay.io は OpenShift Container Platform を設定するほとんどのイメージを提供し、リリースイメージのみが署名されます。リリースイメージは承認済みの OpenShift Container Platform イメージを参照するため、サプライチェーン攻撃からの一定レベルの保護が得られます。ただし、ロギング、モニタリング、サービスメッシュなどの OpenShift Container Platform への拡張の一部は、Operator Lifecycle Manager (OLM) から Operator として提供されます。それらのイメージは、Red Hat Ecosystem Catalog Container イメージ レジストリーから提供されます。

Red Hat レジストリーとインフラストラクチャー間のイメージの整合性を確認するには、署名の検証を有効にします。

2.4.1. Red Hat Container Registry の署名の検証の有効化

Red Hat Container レジストリーのコンテナー署名の検証を有効にするには、これらのレジストリーからのイメージを検証するキーを指定する署名検証ポリシーファイルを作成する必要があります。RHEL8 ノードの場合、レジストリーはデフォルトで **/etc/containers/registries.d** にすでに定義されています。

手順

1. ワーカーノードに必要な設定を含む Butane 設定ファイル **51-worker-rh-registry-trust.bu** を作成します。

   > **注記**
   >
   > 設定ファイルで指定する Butane のバージョン は、OpenShift Container Platform のバージョンと同じである必要があり、末尾は常に **0** です。たとえば、**4.16.0** です。Butane の詳細は、「Butane を使用したマシン設定の作成」を参照してください。

   ```
   variant: openshift
   version: 4.16.0
   ```

```
metadata:
  name: 51-worker-rh-registry-trust
  labels:
    machineconfiguration.openshift.io/role: worker
storage:
  files:
  - path: /etc/containers/policy.json
    mode: 0644
    overwrite: true
    contents:
      inline: |
        {
          "default": [
            {
              "type": "insecureAcceptAnything"
            }
          ],
          "transports": {
            "docker": {
              "registry.access.redhat.com": [
                {
                  "type": "signedBy",
                  "keyType": "GPGKeys",
                  "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
              ],
              "registry.redhat.io": [
                {
                  "type": "signedBy",
                  "keyType": "GPGKeys",
                  "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
              ]
            },
            "docker-daemon": {
              "": [
                {
                  "type": "insecureAcceptAnything"
                }
              ]
            }
          }
        }
```

2. Butane を使用して、ワーカーノードのディスクに書き込まれるファイルが含まれるように、マシン設定 YAML ファイル **51-worker-rh-registry-trust.yaml** を生成します。

```
$ butane 51-worker-rh-registry-trust.bu -o 51-worker-rh-registry-trust.yaml
```

3. 作成されたマシン設定を適用します。

```
$ oc apply -f 51-worker-rh-registry-trust.yaml
```

4. ワーカーマシン設定プールが新しいマシン設定でロールアウトされていることを確認します。

a. 新しいマシン設定が作成されたことを確認します。

```
$ oc get mc
```

出力例

```
NAME                                    GENERATEDBYCONTROLLER
IGNITIONVERSION   AGE
00-master                               a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
00-worker                               a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
01-master-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
01-master-kubelet                          a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
01-worker-container-runtime
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
01-worker-kubelet                          a2178ad522c49ee330b0033bb5cb5ea132060b0a
3.2.0             25m
51-master-rh-registry-trust                                     3.2.0             13s
51-worker-rh-registry-trust                                     3.2.0             53s 1
99-master-generated-crio-seccomp-use-default                                        3.2.0
25m
99-master-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
99-master-ssh                                               3.2.0             28m
99-worker-generated-crio-seccomp-use-default                                        3.2.0
25m
99-worker-generated-registries
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             25m
99-worker-ssh                                               3.2.0             28m
rendered-master-af1e7ff78da0a9c851bab4be2777773b
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             8s
rendered-master-cd51fd0c47e91812bfef2765c52ec7e6
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             24m
rendered-worker-2b52f75684fbc711bd1652dd86fd0b82
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             24m
rendered-worker-be3b3bce4f4aa52a62902304bac9da3c
a2178ad522c49ee330b0033bb5cb5ea132060b0a   3.2.0             48s 2
```

1 新しいマシン設定

2 新しいレンダリングされたマシン設定

b. ワーカーマシン設定プールが新しいマシン設定で更新されていることを確認します。

```
$ oc get mcp
```

出力例

```
NAME     CONFIG                                              UPDATED   UPDATING   DEGRADED
MACHINECOUNT   READYMACHINECOUNT   UPDATEDMACHINECOUNT
```

```
DEGRADEDMACHINECOUNT   AGE
master   rendered-master-af1e7ff78da0a9c851bab4be2777773b   True      False
False     3            3              3                0                    30m
worker   rendered-worker-be3b3bce4f4aa52a62902304bac9da3c   False     True
False     3            0              0                0                    30m (1)
```

1. **UPDATING** フィールドが **True** の場合、マシン設定プールは新しいマシン設定で更新されます。フィールドが **False** になると、ワーカーマシン設定プールが新しいマシン設定にロールアウトされます。

5. クラスターが RHEL7 ワーカーノードを使用している場合、ワーカーマシンの設定プールが更新されたら、それらのノードに YAML ファイルを **/etc/containers/registries.d** ディレクトリーに作成します。これにより、特定のレジストリーサーバーの切り離された署名の場所が指定されます。次の例は、**registry.access.redhat.com** および **registry.redhat.io** でホストされているイメージに対してのみ機能します。

   a. 各 RHEL7 ワーカーノードへのデバッグセッションを開始します。

   ```
   $ oc debug node/<node_name>
   ```

   b. ルートディレクトリーを **/host** に変更します。

   ```
   sh-4.2# chroot /host
   ```

   c. 以下を含む **/etc/containers/registries.d/registry.redhat.io.yaml** ファイルを作成します。

   ```
   docker:
        registry.redhat.io:
           sigstore: https://registry.redhat.io/containers/sigstore
   ```

   d. 以下を含む **/etc/containers/registries.d/registry.access.redhat.com.yaml** ファイルを作成します。

   ```
   docker:
        registry.access.redhat.com:
           sigstore: https://access.redhat.com/webassets/docker/content/sigstore
   ```

   e. デバッグセッションを終了します。

2.4.2. 署名の検証設定の確認

マシン設定をクラスターに適用すると、Machine Config Controller は新規の **MachineConfig** オブジェクトを検出し、新規の **rendered-worker-<hash>** バージョンを生成します。

前提条件

- マシン設定ファイルを使用して署名の検証を有効にしている。

手順

1. コマンドラインで以下のコマンドを実行し、必要なワーカーの情報を表示します。

```
$ oc describe machineconfigpool/worker
```

初期ワーカーモニタリングの出力例

```
Name:         worker
Namespace:
Labels:       machineconfiguration.openshift.io/mco-built-in=
Annotations:  <none>
API Version:  machineconfiguration.openshift.io/v1
Kind:         MachineConfigPool
Metadata:
  Creation Timestamp:  2019-12-19T02:02:12Z
  Generation:          3
  Resource Version:    16229
  Self Link:           /apis/machineconfiguration.openshift.io/v1/machineconfigpools/worker
  UID:                 92697796-2203-11ea-b48c-fa163e3940e5
Spec:
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
    Source:
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         00-worker
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-container-runtime
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         01-worker-kubelet
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         51-worker-rh-registry-trust
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:  machineconfiguration.openshift.io/v1
      Kind:         MachineConfig
      Name:         99-worker-ssh
  Machine Config Selector:
    Match Labels:
      machineconfiguration.openshift.io/role:  worker
  Node Selector:
    Match Labels:
      node-role.kubernetes.io/worker:
  Paused:                              false
Status:
  Conditions:
    Last Transition Time:  2019-12-19T02:03:27Z
    Message:
    Reason:
    Status:                False
    Type:                  RenderDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
```

```
    Status:              False
    Type:                NodeDegraded
    Last Transition Time:  2019-12-19T02:03:43Z
    Message:
    Reason:
    Status:              False
    Type:                Degraded
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:
    Reason:
    Status:              False
    Type:                Updated
    Last Transition Time:  2019-12-19T02:28:23Z
    Message:             All nodes are updating to rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
    Status:              True
    Type:                Updating
  Configuration:
    Name:  rendered-worker-d9b3f4ffcfd65c30dcf591a0e8cf9b2e
    Source:
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 00-worker
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 01-worker-container-runtime
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 01-worker-kubelet
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:          machineconfiguration.openshift.io/v1
      Kind:                 MachineConfig
      Name:                 99-worker-ssh
  Degraded Machine Count:     0
  Machine Count:              1
  Observed Generation:        3
  Ready Machine Count:        0
  Unavailable Machine Count:  1
  Updated Machine Count:      0
Events:                       <none>
```

2. **oc describe** コマンドを再度実行します。

```
$ oc describe machineconfigpool/worker
```

ワーカーの更新後の出力例

```
...
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:             All nodes are updated with rendered-worker-
f6819366eb455a401c42f8d96ab25c02
    Reason:
```

```
    Status:               True
    Type:                 Updated
    Last Transition Time:  2019-12-19T04:53:09Z
    Message:
    Reason:
    Status:               False
    Type:                 Updating
  Configuration:
    Name:  rendered-worker-f6819366eb455a401c42f8d96ab25c02
    Source:
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 00-worker
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 01-worker-container-runtime
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 01-worker-kubelet
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 51-worker-rh-registry-trust
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 99-worker-92697796-2203-11ea-b48c-fa163e3940e5-registries
      API Version:          machineconfiguration.openshift.io/v1
      Kind:               MachineConfig
      Name:                 99-worker-ssh
  Degraded Machine Count:     0
  Machine Count:              3
  Observed Generation:        4
  Ready Machine Count:        3
  Unavailable Machine Count:  0
  Updated Machine Count:      3
...
```

注記

Observed Generation パラメーターは、コントローラーで作成される設定の生成に基づいて増加するカウントを表示します。このコントローラーは、仕様の処理とリビジョンの生成に失敗する場合でも、この値を更新します。**Configuration Source** 値は **51-worker-rh-registry-trust** 設定を参照します。

3. 以下のコマンドを使用して、**policy.json** ファイルが存在することを確認します。

```
$ oc debug node/<node> -- chroot /host cat /etc/containers/policy.json
```

出力例

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
{
  "default": [
```

```
    {
      "type": "insecureAcceptAnything"
    }
  ],
  "transports": {
    "docker": {
      "registry.access.redhat.com": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ],
      "registry.redhat.io": [
        {
          "type": "signedBy",
          "keyType": "GPGKeys",
          "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
        }
      ]
    },
    "docker-daemon": {
      "": [
        {
          "type": "insecureAcceptAnything"
        }
      ]
    }
  }
}
```

4. 以下のコマンドを使用して、**registry.redhat.io.yaml** ファイルが存在することを確認します。

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.redhat.io.yaml
```

出力例

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
docker:
     registry.redhat.io:
         sigstore: https://registry.redhat.io/containers/sigstore
```

5. 以下のコマンドを使用して、**registry.access.redhat.com.yaml** ファイルが存在することを確認します。

```
$ oc debug node/<node> -- chroot /host cat
/etc/containers/registries.d/registry.access.redhat.com.yaml
```

出力例

```
Starting pod/<node>-debug ...
To use host binaries, run `chroot /host`
```

```
docker:
    registry.access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

2.4.3. 検証可能な署名がないコンテナーイメージの検証について

各 OpenShift Container Platform リリースイメージはイミュータブルであり、Red Hat プロダクションキーで署名されています。OpenShift Container Platform の更新またはインストール中に、検証可能な署名がないコンテナーイメージをリリースイメージがデプロイする可能性があります。各署名付きリリースイメージダイジェストはイミュータブルです。リリースイメージ内の各参照は、別のイメージのイミュータブルダイジェストに対するものであるため、コンテンツは推移的に信頼できます。言い換えれば、リリースイメージ上の署名は、すべてのリリース内容を検証します。

たとえば、検証可能な署名がないイメージ参照は、署名付き OpenShift Container Platform リリースイメージに含まれています。

リリース情報の出力例

```
$ oc adm release info quay.io/openshift-release-dev/ocp-release@sha256:2309578b68c5666dad62aed696f1f9d778ae1a089ee461060ba7b9514b7ca417 -o pullspec (1)
quay.io/openshift-release-dev/ocp-v4.0-art-dev@sha256:9aafb914d5d7d0dec4edd800d02f811d7383a7d49e500af548eab5d00c1bffdb (2)
```

1. 署名付きリリースイメージ SHA。
2. リリースに含まれる検証可能な署名がないコンテナーイメージ。

2.4.3.1. 更新時の自動検証

署名の検証は自動的に行われます。OpenShift Cluster Version Operator (CVO) は、OpenShift Container Platform の更新中にリリースイメージの署名を検証します。これは内部プロセスです。自動検証が失敗すると、OpenShift Container Platform のインストールまたは更新は失敗します。

署名の検証は、**skopeo** コマンドラインユーティリティーを使用して手動で行うこともできます。

関連情報

- OpenShift の更新の概要

2.4.3.2. skopeo を使用して Red Hat コンテナーイメージの署名を検証する

OpenShift Container Platform リリースイメージに含まれるコンテナーイメージの署名は、OCP リリースミラーサイト から署名を取得して検証できます。ミラーサイトの署名は Podman や CRI-O が理解できる形式ではないため、**skopeo standalone-verify** コマンドを使用して、リリースイメージが Red Hat によって署名されていることを確認します。

前提条件

- **skopeo** コマンドラインユーティリティーがインストールされている。

手順

1. 次のコマンドを実行して、リリースの完全な SHA を取得します。

```
$ oc adm release info <release_version> \ 1
```

1 <release_version> をリリース番号に置き換えます (例: **4.14.3**)。

出力の抜粋例

```
---
Pull From: quay.io/openshift-release-dev/ocp-
release@sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55
---
```

2. 次のコマンドを実行して、Red Hat リリースキーをプルダウンします。

```
$ curl -o pub.key https://access.redhat.com/security/data/fd431d51.txt
```

3. 次のコマンドを実行して、検証する特定のリリースの署名ファイルを取得します。

```
$ curl -o signature-1 https://mirror.openshift.com/pub/openshift-v4/signatures/openshift-
release-dev/ocp-release/sha256=<sha_from_version>/signature-1 \ 1
```

1 **<sha_from_version>** を、リリースの SHA に一致するミラーサイトへの完全なリンクの SHA 値に置き換えます。たとえば、4.12.23 リリースの署名へのリンクは **https://mirror.openshift.com/pub/openshift-v4/signatures/openshift-release-dev/ocp-release/sha256=e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55/signature-1**、SHA 値は **e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55** です。

4. 次のコマンドを実行して、リリースイメージのマニフェストを取得します。

```
$ skopeo inspect --raw docker://<quay_link_to_release> > manifest.json \ 1
```

1 **<quay_link_to_release>** を、**oc adm release info** コマンドの出力に置き換えます。たとえば、**quay.io/openshift-release-dev/ocp-release@sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55** です。

5. skopeo を使用して署名を検証します。

```
$ skopeo standalone-verify manifest.json quay.io/openshift-release-dev/ocp-release:
<release_number>-<arch> any signature-1 --public-key-file pub.key
```

ここでは、以下のようになります。

<release_number>
: リリース番号を指定します (例: **4.14.3**)。

<arch>
: アーキテクチャーを指定します (例: **x86_64**)。

出力例

```
Signature verified using fingerprint 567E347AD0044ADE55BA8A5F199E2F91FD431D51,
digest sha256:e73ab4b33a9c3ff00c9f800a38d69853ca0c4dfa5a88e3df331f66df8f18ec55
```

2.4.4. 関連情報

- マシン設定の概要

2.5. コンプライアンスについて

多くの OpenShift Container Platform のお客様においては、システムが実稼働環境で使用される前に、一定レベルでの規制への対応またはコンプライアンスが必要になります。この規制対応は、国家標準、業界標準または組織の企業ガバナンスフレームワークによって課せられます。

2.5.1. コンプライアンスおよびリスク管理について

FIPS コンプライアンスは、安全な環境で必要とされる最も重要なコンポーネントの 1 つであり、サポートされている暗号化技術のみがノード上で許可されるようにします。

重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL で FIPS モードを設定する方法の詳細は、RHEL から FIPS モードへの切り替え を参照してください。

FIPS モードでブートされた Red Hat Enterprise Linux (RHEL) または Red Hat Enterprise Linux CoreOS (RHCOS) を実行する場合、OpenShift Container Platform コアコンポーネントは、x86_64、ppc64le、および s390x アーキテクチャーのみで、FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。

OpenShift Container Platform コンプライアンスフレームワークに関する Red Hat のアプローチは、OpenShift セキュリティーガイド のリスク管理および規制対応の章を参照してください。

関連情報

- FIPS モードでのクラスターのインストール

2.6. コンテナーのコンテンツのセキュリティー保護

コンテナー内のコンテンツのセキュリティーを確保するには、まず信頼できるベースイメージ (Red Hat Universal Base Images など) を使用し、信頼できるソフトウェアを追加する必要があります。コンテナーイメージのセキュリティーを継続的に確認するには、Red Hat およびサードパーティーのツールの両方を使用してイメージをスキャンできます。

2.6.1. コンテナー内のセキュリティー

アプリケーションとインフラストラクチャーは、すぐに利用できるコンポーネントで構成されています。その多くは、Linux オペレーティングシステム、JBoss Web Server、PostgreSQL、および Node.js などのオープンソースパッケージです。

これらのパッケージのコンテナー化されたバージョンも利用できます。ただし、パッケージの出所、使用されているバージョン、作成者、またパッケージ内に悪意のあるコードが含まれているかどうかを確認する必要があります。

確認するべき点には以下が含まれます。

- コンテナーの内容がインフラストラクチャーを危険にさらす可能性はあるか？
- アプリケーション層に既知の脆弱性が存在するか？
- ランタイムおよびオペレーティングシステム層は最新の状態にあるか？

Red Hat Universal Base Images (UBI) でコンテナーをビルドすることにより、コンテナーイメージのベースが Red Hat Enterprise Linux に含まれる同じ RPM パッケージのソフトウェアで構成されるものであることを確認できます。UBI イメージの使用または再配布にサブスクリプションは必要ありません。

コンテナー自体のセキュリティーが継続的に保護されるようにするには、RHEL から直接使用されるか、OpenShift Container Platform に追加されているセキュリティースキャン機能は、使用しているイメージに脆弱性がある場合に警告を出します。OpenSCAP イメージスキャンは RHEL で利用でき、Red Hat Quay Container Security Operator は、OpenShift Container Platform で使用されるコンテナーイメージをチェックするために追加できます。

2.6.2. UBI を使用した再配布可能なイメージの作成

コンテナー化されたアプリケーションを作成するには、通常オペレーティングシステムによって提供されるコンポーネントを提供する信頼されたベースイメージの使用から開始します。これらには、ライブラリー、ユーティリティー、およびその他の機能が含まれます。これらは、アプリケーションがオペレーティングシステムのファイルシステムで認識することが予想されます。

Red Hat Universal Base Images (UBI) は、独自のコンテナーをビルドするユーザーに、Red Hat Enterprise Linux rpm パッケージとその他のコンテンツだけで構成されたコンテナーをまずビルドすることを奨励するために作成されました。このような UBI イメージは、セキュリティーパッチを適用し、独自のソフトウェアを組み込むためにビルドされたコンテナーイメージと共に自由に使用し、再配布するために定期的に更新されます。

Red Hat Ecosystem Catalog を検索して、異なる UBI イメージを見つけ、そのイメージの正常性を確認します。セキュアなコンテナーイメージを作成する場合は、以下の 2 つの一般的な UBI イメージのタイプを使用することを検討できるかもしれません。

- **UBI**: RHEL 7、8、9 の標準 UBI イメージ (**ubi7/ubi**、**ubi8/ubi**、**ubi9/ubi**) と、それらのシステムに基づく最小イメージ (**ubi7/ubi-minimal**、**ubi8/ubi-mimimal**、ubi9/ubi-minimal) があります。これらのイメージはすべて、標準の **yum** コマンドおよび **dnf** コマンドを使用して、ビルドするコンテナーイメージに追加できる RHEL ソフトウェアの空きのリポジトリーを参照するように事前に設定されています。

  注記

  Red Hat は、Fedora や Ubuntu などの他のディストリビューションでもこれらのイメージを使用することを推奨しています。

- **Red Hat Software Collections** Red Hat エコシステムカタログで **rhscl/** を検索し、特定タイプのアプリケーションのベースイメージとして使用するために作成されたイメージを見つけます。たとえば、Apache httpd (**rhscl/httpd-***)、Python (**rhscl/python-***)、Ruby (**rhscl/ruby-***)、Node.js (**rhscl/nodejs-***) および Perl (**rhscl/perl-***) rhscl イメージがあります。

UBI イメージは自由に利用でき、再配布可能です。ただし、このイメージに対する Red Hat のサポートは、Red Hat 製品サブスクリプションでのみ利用できることに注意してください。

標準、最小、および init UBI イメージの使用方法とビルド方法については、Red Hat Enterprise Linux ドキュメントの Red Hat Universal Base Images の使用 を参照してください。

2.6.3. RHEL におけるセキュリティースキャン

Red Hat Enterprise Linux (RHEL) システムでは、**openscap-utils** パッケージで OpenSCAP スキャンを利用できます。RHEL では、**openscap-podman** コマンドを使用して、イメージで脆弱性の有無をスキャンできます。Red Hat Enterprise Linux ドキュメントの Scanning containers and container images for vulnerabilities を参照してください。

OpenShift Container Platform では、RHEL スキャナーを CI/CD プロセスで利用することができます。たとえば、ソースコードのセキュリティー上の欠陥をテストする静的コード解析ツールや、既知の脆弱性などのメタデータを提供するために使用するオープンソースライブラリーを特定するソフトウェアコンポジション解析ツールを統合することができます。

2.6.3.1. OpenShift イメージのスキャン

OpenShift Container Platform で実行され、Red Hat Quay レジストリーからプルされるコンテナーイメージの場合、Operator を使用してそれらのイメージの脆弱性を一覧表示できます。Red Hat Quay Container Security Operator を OpenShift Container Platform に追加して、選択した namespace に追加されたイメージの脆弱性レポートを提供できます。

Red Hat Quay のコンテナーイメージスキャンは、Clair によって実行されます。Red Hat Quay では、Clair は RHEL、CentOS、Oracle、Alpine、Debian、および Ubuntu のオペレーティングシステムソフトウェアでビルドされたイメージの脆弱性を検索し、報告することができます。

2.6.4. 外部サービスの統合

OpenShift Container Platform は、オブジェクトのアノテーション (object annotations) を利用して機能を拡張します。脆弱性スキャナーなどの外部ツールはイメージオブジェクトにメタデータのアノテーションを付けることで、結果の要約を表示したり、Pod の実行を制御したりできます。このセクションでは、このアノテーションの認識される形式を説明します。この形式を使用することで、アノテーションをコンソールで安全に使用し、ユーザーに役立つデータを表示することができます。

2.6.4.1. イメージのメタデータ

イメージの品質データには、パッケージの脆弱性およびオープンソースソフトウェア (OSS) ライセンスのコンプライアンスなどの様々なタイプがあります。さらに、複数のプロバイダーがこのメタデータを提供する場合があります。このため、以下のアノテーションの形式が保持されます。

```
quality.images.openshift.io/<qualityType>.<providerId>: {}
```

表2.1 アノテーションキーの形式

コンポーネント	説明	許可される値
qualityType	メタデータのタイプ	**vulnerability** **license** **operations** **policy**

コンポーネント	説明	許可される値
providerId	プロバイダー ID の文字列	**openscap** **redhatcatalog** **redhatinsights** **blackduck** **jfrog**

2.6.4.1.1. アノテーションキーの例

```
quality.images.openshift.io/vulnerability.blackduck: {}
quality.images.openshift.io/vulnerability.jfrog: {}
quality.images.openshift.io/license.blackduck: {}
quality.images.openshift.io/vulnerability.openscap: {}
```

イメージの品質アノテーションの値は、以下の形式に従った構造化データになります。

表2.2 アノテーション値の形式

フィールド	必須？	説明	型
name	はい	プロバイダーの表示名	文字列
timestamp	はい	スキャンのタイムスタンプ	文字列
description	いいえ	簡単な説明	文字列
reference	はい	情報ソースの URL または詳細情報。ユーザーのデータ検証に必要。	文字列
scannerVersion	いいえ	スキャナーバージョン	文字列
compliant	いいえ	コンプライアンスの合否	ブール値
summary	いいえ	検出された問題の要約	リスト (以下の表を参照)

summary フィールドは、以下の形式に従う必要があります。

表2.3 要約フィールド値の形式

フィールド	説明	型

フィールド	説明	型
label	コンポーネントの表示ラベル (例: "critical"、"important"、"moderate"、"low"、または "health")	文字列
data	このコンポーネントのデータ (例: 検出された脆弱性の数またはスコア)	文字列
severityIndex	順序付けおよびグラフィック表示の割り当てを可能にするコンポーネントのインデックス。値は **0..3** の範囲内にあり、**0** = low になります。	整数
reference	情報ソースの URL または詳細情報。オプション:	文字列

2.6.4.1.2. アノテーション値の例

以下の例は、脆弱性の要約データおよびコンプライアンスのブール値を含むイメージの OpenSCAP アノテーションを示しています。

OpenSCAP アノテーション

```
{
  "name": "OpenSCAP",
  "description": "OpenSCAP vulnerability score",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://www.open-scap.org/930492",
  "compliant": true,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "critical", "data": "4", "severityIndex": 3, "reference": null },
    { "label": "important", "data": "12", "severityIndex": 2, "reference": null },
    { "label": "moderate", "data": "8", "severityIndex": 1, "reference": null },
    { "label": "low", "data": "26", "severityIndex": 0, "reference": null }
  ]
}
```

以下の例は、詳細情報として外部 URL と正常性のインデックスデータを含むイメージの Red Hat Ecosystem Catalog のコンテナーイメージのセクション のアノテーションを示しています。

Red Hat Ecosystem Catalog アノテーション

```
{
  "name": "Red Hat Ecosystem Catalog",
  "description": "Container health index",
  "timestamp": "2016-09-08T05:04:46Z",
  "reference": "https://access.redhat.com/errata/RHBA-2016:1566",
```

```
  "compliant": null,
  "scannerVersion": "1.2",
  "summary": [
    { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null }
  ]
}
```

2.6.4.2. イメージオブジェクトのアノテーション

OpenShift Container Platform のエンドユーザーはイメージストリームオブジェクトに対して操作を行いますが、セキュリティーメタデータでアノテーションが付けられるのはイメージオブジェクトです。イメージオブジェクトはクラスター全体でそのスコープが設定され、多くのイメージストリームおよびタグで参照される可能性のある単一イメージをポイントします。

2.6.4.2.1. アノテーションが使用されている CLI コマンドの例

<image> をイメージダイジェストに置き換えます (例: **sha256:401e359e0f45bfdcf004e258b72e253fd07fba8cc5c6f2ed4f4608fb119ecc2**)。

```
$ oc annotate image <image> \
    quality.images.openshift.io/vulnerability.redhatcatalog='{ \
    "name": "Red Hat Ecosystem Catalog", \
    "description": "Container health index", \
    "timestamp": "2020-06-01T05:04:46Z", \
    "compliant": null, \
    "scannerVersion": "1.2", \
    "reference": "https://access.redhat.com/errata/RHBA-2020:2347", \
    "summary": "[ \
      { "label": "Health index", "data": "B", "severityIndex": 1, "reference": null } ]" }'
```

2.6.4.3. Pod 実行の制御

images.openshift.io/deny-execution イメージポリシーを使用して、イメージを実行するかどうかをプログラムで制御します。

2.6.4.3.1. アノテーションの例

```
annotations:
  images.openshift.io/deny-execution: true
```

2.6.4.4. 統合リファレンス

ほとんどの場合、脆弱性スキャナーなどの外部ツールはイメージの更新を監視し、スキャンを実施し、関連するイメージオブジェクトに結果のアノテーションを付けるスクリプトまたはプラグインを開発します。通常、この自動化では OpenShift Container Platform 4.16 REST API を呼び出してアノテーションを作成します。REST API の一般的な情報は、OpenShift Container Platform REST API を参照してください。

2.6.4.4.1. REST API 呼び出しの例

curl を使用する以下の呼び出しの例では、アノテーションの値を上書きします。**<token>**、**<openshift_server>**、**<image_id>**、および **<image_annotation>** の値を置き換えてください。

パッチ API 呼び出し

```
$ curl -X PATCH \
  -H "Authorization: Bearer <token>" \
  -H "Content-Type: application/merge-patch+json" \
  https://<openshift_server>:6443/apis/image.openshift.io/v1/images/<image_id> \
  --data '{ <image_annotation> }'
```

以下は、**PATCH** ペイロードデータの例です。

パッチ呼び出しデータ

```
{
"metadata": {
  "annotations": {
    "quality.images.openshift.io/vulnerability.redhatcatalog":
       "{ 'name': 'Red Hat Ecosystem Catalog', 'description': 'Container health index', 'timestamp': '2020-06-01T05:04:46Z', 'compliant': null, 'reference': 'https://access.redhat.com/errata/RHBA-2020:2347', 'summary': [{'label': 'Health index', 'data': '4', 'severityIndex': 1, 'reference': null}] }"
    }
  }
}
```

関連情報

- イメージストリームオブジェクト

2.7. コンテナーレジストリーのセキュアな使用

コンテナーレジストリーは、以下を実行するためにコンテナーイメージを保存します。

- イメージに他からアクセスできるようにする
- イメージをイメージの複数バージョンを含むことができるリポジトリーに整理する
- オプションで、異なる認証方法に基づいてイメージへのアクセスを制限するか、イメージを一般に利用できるようにする。

Quay.io や Docker Hub などのパブリックコンテナーレジストリーがあり、ここでは多くの人や組織がイメージを共有します。Red Hat レジストリーは、サポート対象の Red Hat およびパートナーのイメージを提供しますが、Red Hat Ecosystem Catalog ではこれらのイメージに関する詳細な説明およびヘルスチェックが提供されます。独自のレジストリーを管理するには、Red Hat Quay などのコンテナーレジストリーを購入することができます。

セキュリティーの観点では、一部のレジストリーは、コンテナーの正常性を確認し、強化するために特別な機能を提供します。たとえば、Red Hat Quay は、Clair セキュリティースキャナーを使用したコンテナー脆弱性のスキャン、GitHub およびその他の場所でソースコードが変更された場合にイメージを自動的に再ビルドするためのビルドのトリガー、およびイメージへのアクセスをセキュア化するためのロールベースのアクセス制御 (RBAC) を使用できる機能を提供します。

2.7.1. コンテナーのソースの確認

ダウンロード済みかつデプロイ済みのコンテナーイメージのコンテンツをスキャンし、追跡するには各種のツールを使用できます。しかし、コンテナーイメージの公開ソースは数多くあります。公開されて

いるコンテナーレジストリーを使用する場合は、信頼されるソースを使用して保護用の層を追加することができます。

2.7.2. イミュータブルで認定済みのコンテナー

イミュータブルなコンテナー を管理する際に、セキュリティー更新を使用することはとくに重要になります。イミュータブルなコンテナーは、実行中には変更されることのないコンテナーです。イミュータブルなコンテナーをデプロイする場合には、実行中のコンテナーにステップインして 1 つ以上のバイナリーを置き換えることはできません。運用上の観点では、更新されたコンテナーイメージを再ビルド、再デプロイし、コンテナーを変更するのではなく、コンテナーの置き換えを行います。

以下は、Red Hat 認定イメージの特徴になります。

- プラットフォームの各種コンポーネントまたは層に既知の脆弱性がない。
- ベアメタルからクラウドまで、RHEL プラットフォーム全体で互換性がある。
- Red Hat によってサポートされる。

既知の脆弱性のリストは常に更新されるので、デプロイ済みのコンテナーイメージのコンテンツのほか、新規にダウンロードしたイメージを継続的に追跡する必要があります。Red Hat セキュリティーアドバイザリー (RHSA) を利用して、Red Hat 認定コンテナーイメージで新たに発見される問題に関する警告を受け、更新されたイメージを確認することができます。または、Red Hat Ecosystem Catalog にアクセスして、その問題および各 Red Hat イメージの他のセキュリティー関連の問題を検索することもできます。

2.7.3. Red Hat レジストリーおよび Ecosystem Catalog からのコンテナーの取得

Red Hat では、Red Hat Ecosystem Catalog の Container Images セクションから、Red Hat 製品およびパートナーオファリングの認定コンテナーイメージをリスト表示しています。このカタログから、CVE、ソフトウェアパッケージのリスト、ヘルススコアなどの各イメージの詳細を確認できます。

Red Hat イメージは、パブリックコンテナーレジストリー (**registry.access.redhat.com**) および認証されたレジストリー (**registry.redhat.io**) によって代表される、**Red Hat レジストリー** というレジストリーに実際に保存されます。どちらにも基本的に Red Hat サブスクリプション認証情報での認証を必要とするいくつかの追加イメージを含む **registry.redhat.io** と同様に、同じコンテナーイメージのセットが含まれます。

Red Hat ではコンテナーのコンテンツの脆弱性を監視し、コンテンツを定期的に更新しています。**glibc**、DROWN、または Dirty Cow の修正など、Red Hat がセキュリティー更新をリリースする際に、影響を受けるすべてのコンテナーイメージも再ビルドされ、Red Hat Registry にプッシュされます。

Red Hat では **health index** を使用して、Red Hat Ecosystem Catalog 経由で提供される各コンテナーのセキュリティー上のリスクを考慮します。コンテナーは Red Hat およびエラータプロセスで提供されるソフトウェアを使用するため、セキュリティーのレベルは、コンテナーが古いと低くなり、新規のコンテナーの場合はセキュリティーのレベルが上がります。

コンテナーの年数について、Red Hat Ecosystem Catalog では格付けシステムを使用します。最新度の評価は、イメージに利用できる最も古く、最も重大度の高いセキュリティーエラータに基づいて行われます。格付けは "A" から "F" まであり、"A" が最新となります。この格付けシステムの詳細は、Container Health Index grades as used inside the Red Hat Ecosystem Catalog を参照してください。

Red Hat ソフトウェアに関連するセキュリティー更新および脆弱性の詳細は、Red Hat Product Security Center を参照してください。Red Hat セキュリティーアドバイザリー を参照して、特定のアドバイザリーおよび CVE を検索できます。

2.7.4. OpenShift Container レジストリー

OpenShift Container Platform には、コンテナーイメージを管理するために使用できるプラットフォームの統合されたコンポーネントとして実行される、プライベートレジストリーの **OpenShift Container レジストリー** が含まれます。OpenShift Container レジストリーは、ロールベースのアクセス制御を提供します。これにより、どのコンテナーイメージを誰がプル/プッシュするのかを管理できるようになります。

また、OpenShift Container Platform は Red Hat Quay などのすでに使用している可能性のある他のプライベートレジストリーとの統合もサポートしています。

関連情報

- 統合 OpenShift イメージレジストリー

2.7.5. Red Hat Quay を使用したコンテナーの保存

Red Hat Quay は、Red Hat のエンタープライズレベルの品質の高いコンテナーレジストリー製品です。Red Hat Quay の開発は、アップストリームの Project Quay で行われます。Red Hat Quay は、オンプレミスまたは Quay.io のホスト型バージョンの Red Hat Quay でデプロイできます。

Red Hat Quay のセキュリティー関連機能には、以下が含まれます。

- **Time machine**: 古いタグが付いたイメージを、一定期間後またはユーザーが選択した有効期限に基づいて期限切れにすることができます。
- **Repository mirroring** (リポジトリーのミラーリング): セキュリティー上の理由から他のレジストリーをミラーリングします。たとえば、会社のファイアウォールの背後の Red Hat Quay でパブリックリポジトリーをホストしたり、パフォーマンス上の理由からレジストリーを使用される場所の近くに配置したりします。
- **Action log storage** (アクションログの保存): Red Hat Quay のロギング出力を Elasticsearch ストレージまたは Splunk に保存し、後に検索および分析に使用できるようにします。
- **Clair**: 各コンテナーイメージの起点に基づき、さまざまな Linux 脆弱性データベースに対してイメージをスキャンします。
- **Internal authentication** (内部認証): Red Hat Quay への RBAC 認証を処理するデフォルトのローカルデータベースを使用するか、LDAP、Keystone (OpenStack)、JWT Custom Authentication、または External Application Token 認証から選択します。
- **External authorization (OAuth)** (外部認証 (OAuth): GitHub、GitHub Enterprise、または Google 認証からの Red Hat Quay への認証を許可します。
- **Access settings** (アクセス設定): docker、rkt、匿名アクセス、ユーザー作成のアカウント、暗号化されたクライアントパスワード、または接頭辞、ユーザー名の自動補完での Red Hat Quay へのアクセスを可能にするトークンを生成します。

Red Hat Quay と OpenShift Container Platform の統合が継続されており、とくに関連する OpenShift Container Platform Operator との統合が継続されています。Quay Bridge Operator を使用すると、内部 OpenShift イメージレジストリーを Red Hat Quay に置き換えることができます。Red Hat Quay

Container Security Operator を使用すると、Red Hat Quay レジストリーからプルされた OpenShift Container Platform で実行されているイメージの脆弱性を確認することができます。

2.8. ビルドプロセスのセキュリティー保護

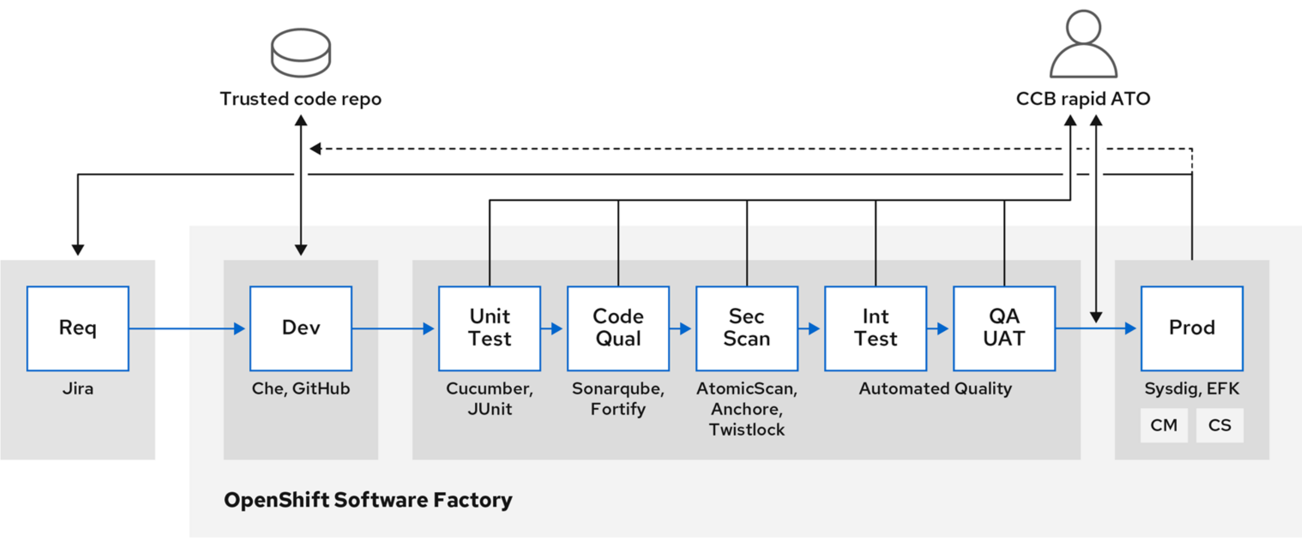
コンテナー環境では、ソフトウェアのビルドプロセスはライフサイクルのステージであり、ここでは、アプリケーションコードが必要なランタイムライブラリーと統合されます。このビルドプロセスの管理は、ソフトウェアのスタックのセキュリティーを保護する上で鍵となります。

2.8.1. 1 回のビルドでどこにでもデプロイが可能

OpenShift Container Platform をコンテナービルドの標準プラットフォームとして使用することで、ビルド環境のセキュリティーを確保できます。「1 回のビルドでどこにでもデプロイが可能」という理念を背景に、ビルドプロセスの製品がそのままの状態で実稼働にデプロイされるようにすることができます。

コンテナーのイミュータブルな状態を維持することも重要です。実行中のコンテナーにパッチを当てることはできません。その代わりに再ビルドおよび再デプロイを実行します。

ソフトウェアがビルド、テスト、および実稼働環境の複数ステージを通過する際に、ソフトウェアのサプライチェーンを設定するツールが信頼できるかどうかは重要です。以下の図は、コンテナー化されたソフトウェアの信頼できるソフトウェアサプライチェーンに組み込むことができるプロセスおよびツールを示しています。



OpenShift Container Platform は、セキュアなコードを作成し、管理できるように、信頼できるコードリポジトリー (GitHub など) および開発プラットフォーム (Che など) と統合できます。単体テストは、Cucumber および JUnit に依存する必要がある場合があります。

Red Hat Advanced Cluster Security for Kubernetes を使用すると、ビルド、デプロイ、または実行時にコンテナーの脆弱性や設定の問題を検査できます。Quay に保存されているイメージについては、Clair スキャナー を使用して保存中のイメージを検査できます。さらに、Red Hat Ecosystem Catalog で 認定済みの脆弱性スキャナー が提供されています。

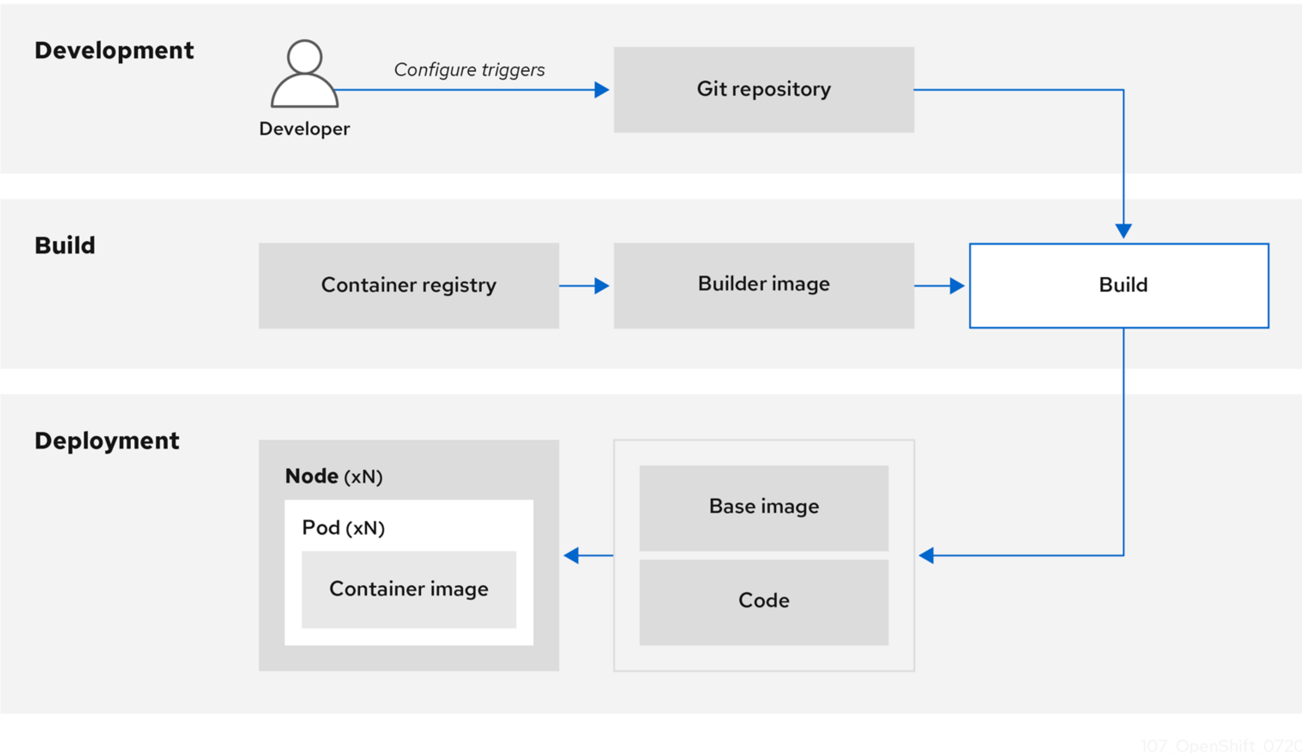
Sysdig などのツールを使用すると、コンテナー化されたアプリケーションを継続的に監視できます。

2.8.2. ビルドの管理

Source-to-Image (S2I) を使用して、ソースコードとベースイメージを組み合わせることができます。**ビルダーイメージ** は S2I を利用し、開発および運用チームの再現可能なビルド環境での協業を可能にします。Red Hat S2I イメージが Universal Base Image (UBI) イメージとして利用可能な場合、実際の RHEL RPM パッケージからビルドされたベースイメージでソフトウェアを自由に再配布できます。Red Hat は、これを可能にするためにサブスクリプションの制限を削除しました。

開発者がビルドイメージを使用して、アプリケーション用に Git でコードをコミットする場合、OpenShift Container Platform は以下の機能を実行できます。

- コードリポジトリーの Webhook または他の自動化された継続的インテグレーション (CI) プロセスのいずれかで、利用可能なアーティファクト、S2I ビルダーイメージ、および新たにコミットされたコードを使用して新規イメージの自動アセンブルをトリガーします。
- 新規にビルドしたイメージを自動的にデプロイし、テストします。
- テスト済みのイメージを実稼働にプロモートします。ここでは CI プロセスを使用して自動的にデプロイされます。



統合された OpenShift Container レジストリーを使用して、最終イメージへのアクセスを管理できます。S2I イメージおよびネイティブビルドイメージの両方は OpenShift Container レジストリーに自動的にプッシュされます。

CI の組み込まれた Jenkins のほかに、独自のビルドおよび CI 環境を RESTful API および API 準拠のイメージレジストリーを使用して OpenShift Container Platform に統合することもできます。

2.8.3. ビルド時の入力のセキュリティー保護

シナリオによっては、ビルド操作において、依存するリソースにアクセスするために認証情報が必要になる場合がありますが、この認証情報をビルドで生成される最終的なアプリケーションイメージで利用可能にすることは適切ではありません。このため、入力シークレットを定義することができます。

たとえば、Node.js アプリケーションのビルド時に、Node.js モジュールのプライベートミラーを設定できます。プライベートミラーからモジュールをダウンロードするには、URL、ユーザー名、パスワード

を含む、ビルド用のカスタム **.npmrc** ファイルを指定する必要があります。セキュリティー上の理由により、認証情報はアプリケーションイメージで公開しないでください。

このサンプルシナリオを使用すると、新しい **BuildConfig** オブジェクトに入力シークレットを追加できます。

手順

1. シークレットがない場合は作成します。

```
$ oc create secret generic secret-npmrc --from-file=.npmrc=~/.npmrc
```

   これにより、**secret-npmrc** という名前の新規シークレットが作成されます。これには、~/**.npmrc** ファイルの base64 でエンコードされたコンテンツが含まれます。

2. シークレットを既存の **BuildConfig** オブジェクトの **source** セクションに追加します。

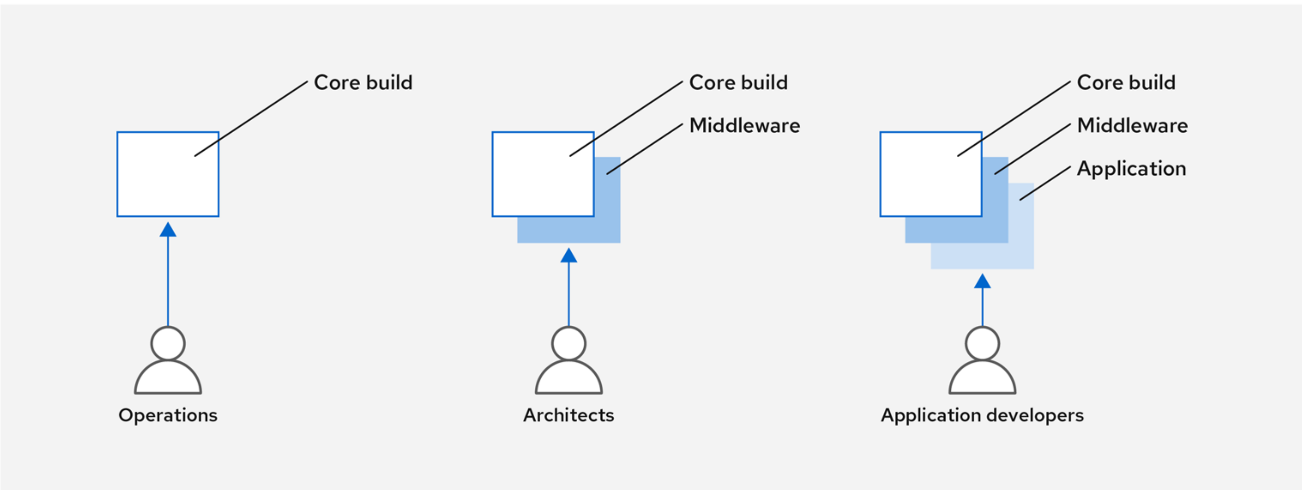
```
source:
  git:
    uri: https://github.com/sclorg/nodejs-ex.git
  secrets:
  - destinationDir: .
    secret:
      name: secret-npmrc
```

3. シークレットを新規の **BuildConfig** オブジェクトに追加するには、以下のコマンドを実行します。

```
$ oc new-build \
    openshift/nodejs-010-centos7~https://github.com/sclorg/nodejs-ex.git \
    --build-secret secret-npmrc
```

2.8.4. ビルドプロセスの設計

コンテナーの層を使用できるようにコンテナーイメージ管理およびビルドプロセスを設計して、制御を分離可能にすることができます。



たとえば、運用チームはベースイメージを管理します。一方で、アーキテクトはミドルウェア、ランタイム、データベース、その他のソリューションを管理します。これにより、開発者はアプリケーション層のみを使用し、コードの作成に集中することができます。

新しい脆弱性情報は常に更新されるので、コンテナーのコンテンツを継続的かつプロアクティブに確認する必要があります。これを実行するには、自動化されたセキュリティーテストをビルドまたは CI プロセスに統合する必要があります。以下に例を示します。

- SAST / DAST – 静的および動的なセキュリティーテストツール
- 既知の脆弱性をリアルタイムにチェックするためのスキャナー。このようなツールは、コンテナー内のオープンソースパッケージをカタログ化し、既知の脆弱性を通知し、スキャン済みのパッケージに新たな脆弱性が検出されるとその更新情報を送信します。

CI プロセスには、セキュリティースキャンで発見される問題を担当チームが適切に対処できるように、これらの問題のフラグをビルドに付けるポリシーを含める必要があります。カスタマイズしたコンテナーに署名することで、ビルドとデプロイメント間に改ざんが発生しないようにします。

GitOps の方法を使用すると、同じ CI/CD メカニズムを使用してアプリケーションの設定だけでなく、OpenShift Container Platform インフラストラクチャーも管理できます。

2.8.5. Knative サーバーレスアプリケーションのビルド

Kubernetes と Kourier を利用すると、OpenShift Container Platform で OpenShift Serverless を使用して、サーバーレスアプリケーションを構築、デプロイ、管理できます。

他のビルドと同様に、S2I イメージを使用してコンテナーをビルドしてから、Knative サービスを使用してそれらを提供できます。OpenShift Container Platform Web コンソールの **Topology** ビューを使用して Knative アプリケーションのビルドを表示します。

2.8.6. 関連情報

- イメージビルドについて
- ビルドのトリガーおよび変更
- ビルド入力の作成
- 入力シークレットおよび config map
- OpenShift Serverless の概要
- Topology ビューを使用したアプリケーション構成の表示

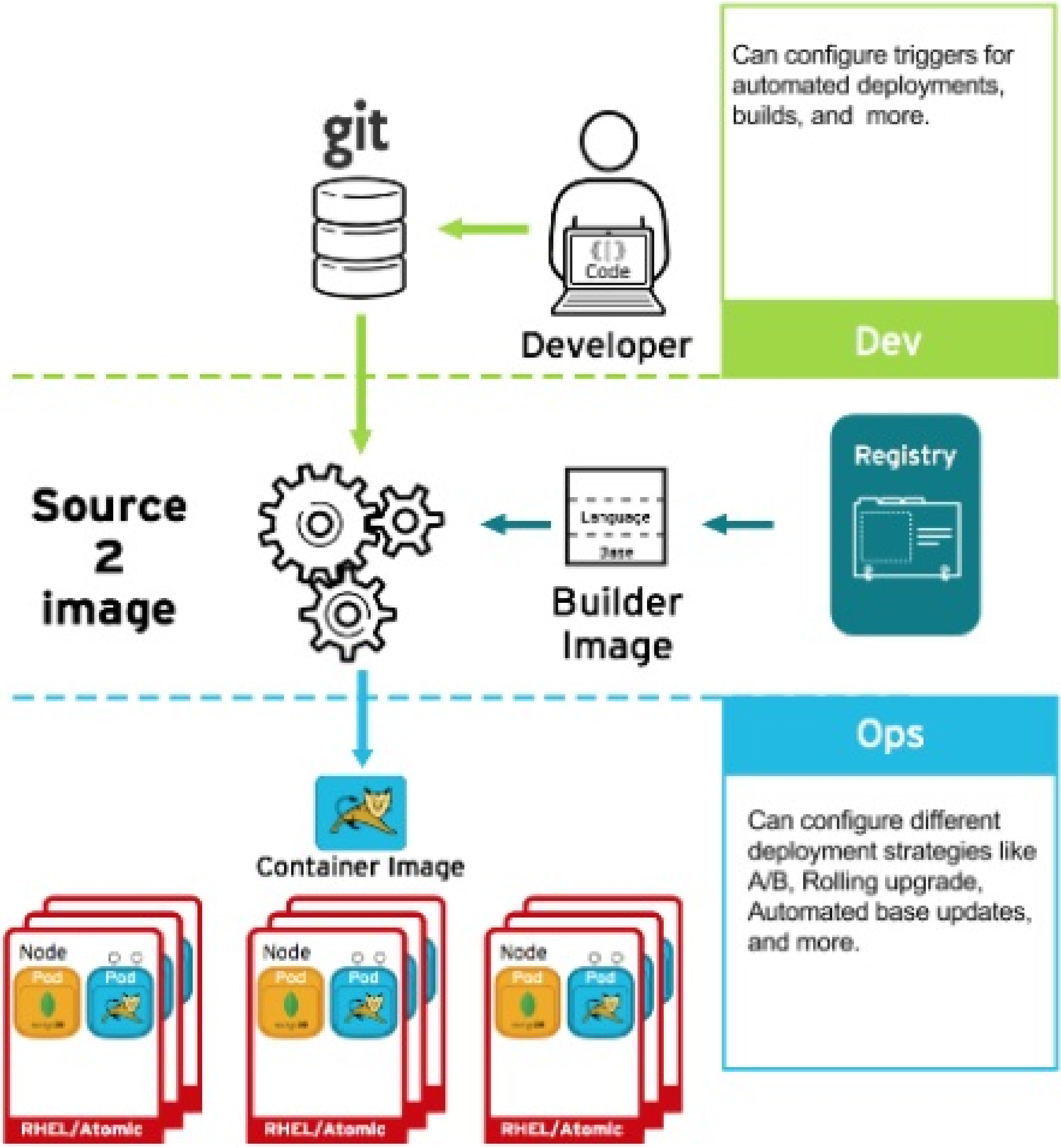
2.9. コンテナーのデプロイ

各種の手法を使用して、デプロイするコンテナーが最新の実稼働に適した品質のコンテンツを保持し、改ざんされていないことを確認することができます。これらの手法には、最新のコードを組み込むためのビルドトリガーのセットアップやコンテナーが信頼できるソースから取得され、変更されないようにするための署名の使用が含まれます。

2.9.1. トリガーによるコンテナーデプロイメントの制御

ビルドプロセスで何らかの問題が生じる場合や、イメージのデプロイ後に脆弱性が発見される場合に、自動化されるポリシーベースのデプロイのためのツールを使用して修復できます。イメージの再ビルド

および置き換えはトリガーを使用して実行し、イミュータブルなコンテナーのプロセスを確認できます。実行中のコンテナーにパッチを当てる方法は推奨されていません。



たとえば、3 つのコンテナーイメージ層 (コア、ミドルウェア、アプリケーション) を使用してアプリケーションをビルドするとします。コアイメージに問題が見つかり、そのイメージは再ビルドされました。ビルドが完了すると、イメージは OpenShift Container レジストリーにプッシュされます。OpenShift Container Platform はイメージが変更されたことを検知し、定義されたトリガーに基づいてアプリケーションイメージを自動的に再ビルドし、デプロイします。この変更には修正されたライブラリーが組み込まれ、実稼働コードが最新のイメージと同じ状態になります。

oc set triggers コマンドを使用してデプロイメントトリガーを設定できます。たとえば、deployment-example という名前のデプロイメントのトリガーを設定するには、以下を実行します。

```
$ oc set triggers deploy/deployment-example \
    --from-image=example:latest \
    --containers=web
```

2.9.2. イメージソースのデプロイの制御

重要な点として、対象とするイメージが実際にデプロイされていることや、組み込まれているコンテンツを持つイメージが信頼されるソースからのものであること、またそれらが変更されていないことを確認できる必要があります。これは、暗号による署名を使用して実行できます。OpenShift Container Platform では、クラスター管理者がデプロイメント環境とセキュリティー要件を反映した (広義または狭義のものを含む) セキュリティーポリシーを適用できます。このポリシーは、以下の 2 つのパラメーターで定義されます。

- 1 つ以上のレジストリー (オプションのプロジェクト namespace を使用)
- 信頼タイプ (accept、reject、または require public key(s))

これらのポリシーパラメーターを使用して、レジストリー全体、レジストリーの一部、または個別のイメージに対して信頼関係を許可、拒否、または要求することができます。信頼されたパブリックキーを使用して、ソースが暗号で検証されていることを確認できます。このポリシールールはノードに適用されます。ポリシーは、すべてのノード全体に均一に適用されるか、異なるノードのワークロード (例: ビルド、ゾーン、または環境) ごとにターゲットが設定される場合があります。

イメージ署名ポリシーファイルの例

```
{
    "default": [{"type": "reject"}],
    "transports": {
        "docker": {
            "access.redhat.com": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ]
        },
        "atomic": {
            "172.30.1.1:5000/openshift": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/rpm-gpg/RPM-GPG-KEY-redhat-release"
                }
            ],
            "172.30.1.1:5000/production": [
                {
                    "type": "signedBy",
                    "keyType": "GPGKeys",
                    "keyPath": "/etc/pki/example.com/pubkey"
                }
            ],
            "172.30.1.1:5000": [{"type": "reject"}]
        }
    }
}
```

ポリシーは **/etc/containers/policy.json** としてノードに保存できます。このファイルのノードへの保存は、新規の **MachineConfig** オブジェクトを使用して実行するのが最適な方法です。この例では、以下のルールを実施しています。

- Red Hat レジストリー (**registry.access.redhat.com**) からのイメージは Red Hat パブリックキーで署名される必要がある。
- **openshift** namespace 内の OpenShift Container レジストリーからのイメージは Red Hat パブリックキーで署名される必要がある。
- **production** namespace 内の OpenShift Container レジストリーからのイメージは **example.com** のパブリックキーで署名される必要がある。
- グローバルの **default** 定義で指定されていないその他すべてのレジストリーは拒否される。

2.9.3. 署名トランスポートの使用

署名トランスポートは、バイナリーの署名 Blob を保存および取得する方法です。署名トランスポートには、2 つのタイプあります。

- **atomic**: OpenShift Container Platform API で管理される。
- **docker**: ローカルファイルとして提供されるか、Web サーバーによって提供される。

OpenShift Container Platform API は、**atomic** トランスポートタイプを使用する署名を管理します。このタイプの署名を使用するイメージは OpenShift Container レジストリーに保存する必要があります。docker/distribution **extensions** API はイメージ署名のエンドポイントを自動検出するため、追加の設定は不要になります。

docker トランスポートタイプを使用する署名は、ローカルファイルまたは Web サーバーによって提供されます。これらの署名には柔軟性があります。任意のコンテナーレジストリーからイメージを提供でき、バイナリー署名の送信に個別のサーバーを使用することができます。

ただし、**docker** トランスポートタイプの場合には追加の設定が必要です。任意に名前が付けられた YAML ファイルをホストシステムのディレクトリー (**/etc/containers/registries.d**) にデフォルトとして配置し、ノードを署名サーバーの URI で設定する必要があります。YAML 設定ファイルには、レジストリー URI および署名サーバー URI が含まれます。署名サーバー URI は、**sigstore** とも呼ばれます。

registries.d ファイルの例

```
docker:
    access.redhat.com:
        sigstore: https://access.redhat.com/webassets/docker/content/sigstore
```

この例では、Red Hat レジストリー (**access.redhat.com**) は、**docker** タイプのトランスポートの署名を提供する署名サーバーです。Red Hat レジストリーの URI は、**sigstore** パラメーターで定義されます。このファイルに **/etc/containers/registries.d/redhat.com.yaml** という名前を付け、Machine Config Operator を使用してこのファイルをクラスター内の各ノード上に自動的に配置することができます。ポリシーと **registries.d** ファイルはコンテナーのランタイムで動的に読み込まれるため、サービスを再起動する必要はありません。

2.9.4. シークレットおよび config map の作成

Secret オブジェクトタイプはパスワード、OpenShift Container Platform クライアント設定ファイル、**dockercfg** ファイル、プライベートソースリポジトリーの認証情報などの機密情報を保持するメカニズムを提供します。シークレットは機密内容を Pod から切り離します。シークレットはボリュームプラグインを使用してコンテナーにマウントすることも、システムが Pod の代わりにシークレットを使用して各種アクションを実行することもできます。

たとえば、プライベートイメージリポジトリーにアクセスできるように、シークレットをデプロイメント設定に追加するには、以下を実行します。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. 新規プロジェクトを作成します。
3. **Resources** → **Secrets** に移動し、新規シークレットを作成します。**Secret Type** を **Image Secret** に、**Authentication Type** を **Image Registry Credentials** に設定し、プライベートイメージリポジトリーにアクセスするために必要な認証情報を入力します。
4. デプロイメント設定を作成する場合 (例: **Add to Project** → **Deploy Image** ページに移動する)、**Pull Secret** を新規シークレットに設定します。

config map はシークレットに似ていますが、機密情報を含まない文字列の使用をサポートするように設計されています。**ConfigMap** オブジェクトは、Pod で使用したり、コントローラーなどのシステムコンポーネントの設定データを保存するために使用できる設定データのキーと値のペアを保持します。

2.9.5. 継続的デプロイメントの自動化

独自の継続的デプロイメント (CD) のツールを OpenShift Container Platform に統合することができます。

CI/CD および OpenShift Container Platform を利用することで、アプリケーションの再ビルドプロセスを自動化し、最新の修正の組み込み、テスト、および環境内の至るところでのデプロイを可能にします。

関連情報

- 入力シークレットおよび config map

2.10. コンテナープラットフォームのセキュリティー保護

OpenShift Container Platform および Kubernetes API は、スケーリング時にコンテナー管理を自動化する鍵となります。API は以下の目的で使用されます。

- Pod、サービス、およびレプリケーションコントローラーのデータの検証および設定。
- 受信要求におけるプロジェクト検証の実施と、他の主要なシステムコンポーネントでのトリガーの呼び出し。

Kubernetes をベースとする OpenShift Container Platform のセキュリティー関連機能には、以下が含まれます。

- マルチテナンシー: ロールベースのアクセス制御とネットワークポリシーを統合し、複数のレベルでコンテナーを分離します。
- API と API の要求側との間の境界を形成する受付プラグイン。

OpenShift Container Platform は Operator を使用して Kubernetes レベルのセキュリティー機能の管理を自動化し、単純化します。

2.10.1. マルチテナンシーによるコンテナーの分離

マルチテナンシーは、複数のユーザーによって所有され、複数のホストおよび namespace で実行される OpenShift Container Platform クラスターの複数アプリケーションが、相互に分離された状態のままにし、外部の攻撃から隔離された状態にすることができます。ロールベースアクセス制御 (RBAC) を Kubernetes namespace に適用して、マルチテナンシーを取得します。

Kubernetes では、**namespace** はアプリケーションを他のアプリケーションと分離した状態で実行できるエリアです。OpenShift Container Platform は、SELinux の MCS ラベルを含む追加のアノテーションを追加して namespace を使用し、これを拡張し、これらの拡張された namespace を **プロジェクト** として特定します。プロジェクトの範囲内で、ユーザーは、サービスアカウント、ポリシー、制約、およびその他のオブジェクトなど、独自のクラスターリソースを維持できます。

RBAC オブジェクトはプロジェクトに割り当てられ、選択されたユーザーのそれらのプロジェクトへのアクセスを認可します。この認可には、ルール、ロール、およびバインディングの形式が使用されます。

- ルールは、ユーザーがプロジェクト内で作成またはアクセスできるものを定義します。
- ロールは、選択されたユーザーまたはグループにバインドできるルールのコレクションです。
- バインディングは、ユーザーまたはグループとロール間の関連付けを定義します。

ローカル RBAC ロールおよびバインディングは、ユーザーまたはグループを特定のプロジェクトに割り当てます。クラスター RBAC では、クラスター全体のロールおよびバインディングをクラスターのすべてのプロジェクトに割り当てることができます。**admin**、**basic-user**、**cluster-admin**、および **cluster-status** アクセスを提供するために割り当てることのできるデフォルトのクラスターロールがあります。

2.10.2. 受付プラグインでのコントロールプレーンの保護

RBAC はユーザーおよびグループと利用可能なプロジェクト間のアクセスルールを制御しますが、**受付プラグイン** は OpenShift Container Platform マスター API へのアクセスを定義します。受付プラグインは、以下で構成されるルールのチェーンを形成します。

- デフォルトの受付プラグイン: これは、OpenShift Container Platform コントロールプレーンのコンポーネントに適用されるポリシーおよびリソース制限のデフォルトセットを実装します。
- ミューティングアドミッションプラグイン: これらのプラグインは、アドミッションチェーンを動的に拡張します。これらは Webhook サーバーに対する呼び出しを実行し、要求の認証および選択されたリソースの変更の両方を実行します。
- 検証用の受付プラグイン: 選択されたリソースの要求を検証し、要求を検証すると共にリソースが再度変更されないようにすることができます。

API 要求はチェーン内の受付プラグインを通過し、途中で失敗した場合には要求が拒否されます。それぞれの受付プラグインは特定のリソースに関連付けられ、それらのリソースの要求にのみ応答します。

2.10.2.1. Security Context Constraints (SCC)

Security Context Constraints (SCC) を使用して、Pod のシステムでの受け入れを可能にするために Pod の実行時に必要となる一連の条件を定義することができます。

以下は、SCC で管理できる分野の一部です。

- 特権付きコンテナーの実行
- コンテナーが要求できる機能の追加

- ホストディレクトリーのボリュームとしての使用
- コンテナーの SELinux コンテキスト
- コンテナーのユーザー ID

必要なパーミッションがある場合は、必要に応じてデフォルトの SCC ポリシーの許容度を上げるように調整することができます。

2.10.2.2. ロールのサービスアカウントへの付与

ロールは、ユーザーにロールベースのアクセスを割り当てるのと同じ方法で、サービスアカウントに割り当てることができます。各プロジェクトに 3 つのデフォルトサービスアカウントが作成されます。サービスアカウント:

- スコープが特定プロジェクトに制限される
- その名前はそのプロジェクトから派生している。
- OpenShift Container レジストリーにアクセスするために API トークンおよび認証情報が自動的に割り当てられる。

プラットフォームのコンポーネントに関連付けられたサービスアカウントでは、キーが自動的にローテーションされます。

2.10.3. 認証および認可

2.10.3.1. OAuth を使用したアクセスの制御

コンテナープラットフォームのセキュリティーを保護にするために、認証および承認で API アクセス制御を使用することができます。OpenShift Container Platform マスターには、ビルトインの OAuth サーバーが含まれます。ユーザーは、OAuth アクセストークンを取得して API に対して認証することができます。

管理者として、LDAP、GitHub、または Google などの **アイデンティティープロバイダー** を使用して認証できるように OAuth を設定できます。新規の OpenShift Container Platform デプロイメントには、デフォルトでアイデンティティープロバイダーが使用されますが、これを初回インストール時またはインストール後に設定できます。

2.10.3.2. API アクセス制御および管理

アプリケーションには、管理を必要とする各種のエンドポイントを持つ複数の独立した API サービスを設定できます。OpenShift Container Platform には 3scale API ゲートウェイのコンテナー化されたバージョンが含まれており、これにより API を管理し、アクセスを制御することができます。

3scale は、API の認証およびセキュリティーに関する様々な標準オプションを提供します。これらは、認証情報を発行し、アクセスを制御するために単独で使用することも、他と組み合わせて使用することもできます (例: 標準 API キー、アプリケーション ID とキーペア、OAuth 2.0 など)。

アクセスは、特定のエンドポイント、メソッド、およびサービスに制限することができ、アクセスポリシーをユーザーグループに適用することができます。アプリケーションの計画に基づいて、API の使用にレート制限を設定したり、開発者グループのトラフィックフローを制御したりすることが可能です。

APIcast v2 (コンテナー化された 3scale API ゲートウェイ) の使用に関するチュートリアルは、3scale ドキュメントの Running APIcast on Red Hat OpenShift を参照してください。

2.10.3.3. Red Hat Single Sign-On

Red Hat Single Sign-On サーバーを使用すると、SAML 2.0、OpenID Connect、および OAuth 2.0 などの標準に基づく Web サインオン機能を提供し、アプリケーションのセキュリティーを保護することができます。このサーバーは、SAML または OpenID Connect ベースのアイデンティティープロバイダー (IdP) として機能します。つまり、標準ベースのトークンを使用して、アイデンティティー情報およびアプリケーションについてエンタープライズユーザーディレクトリーまたはサードパーティーのアイデンティティープロバイダーとの仲介を行います。Red Hat Single Sign-On を Microsoft Active Directory および Red Hat Enterprise Linux Identity Management を含む LDAP ベースのディレクトリーサービスと統合することが可能です。

2.10.3.4. セルフサービス Web コンソールのセキュリティー保護

OpenShift Container Platform はセルフサービスの Web コンソールを提供して、チームが認証なしに他の環境にアクセスできないようにします。OpenShift Container Platform は以下の条件に基づいてセキュアなマルチテナントマスターを提供します。

- マスターへのアクセスは Transport Layer Security (TLS) を使用する。
- API サーバーへのアクセスは X.509 証明書または OAuth アクセストークンを使用する。
- プロジェクトのクォータは不正トークンによるダメージを制限する。
- etcd サービスはクラスターに直接公開されない。

2.10.4. プラットフォームの証明書の管理

OpenShift Container Platform には、そのフレームワーク内に、TLS 証明書による暗号化を利用した REST ベースの HTTPS 通信を使用する複数のコンポーネントがあります。OpenShift Container Platform のインストーラーは、これらの認証をインストール時に設定します。以下は、このトラフィックを生成するいくつかの主要コンポーネントです。

- マスター (API サーバーとコントローラー)
- etcd
- ノード
- レジストリー
- ルーター

2.10.4.1. カスタム証明書の設定

API サーバーおよび Web コンソールのパブリックホスト名のカスタム提供証明書は、初回のインストール時または証明書の再デプロイ時に設定できます。カスタム CA を使用することも可能です。

関連情報

- OpenShift Container Platform の紹介
- RBAC の使用によるパーミッションの定義および適用
- 受付プラグインについて
- Security Context Constraints の管理

- SCC リファレンスコマンド
- ロールをサービスアカウントに付与する例
- 内部 OAuth サーバーの設定
- アイデンティティープロバイダー設定について
- 証明書の種類および説明
- プロキシー証明書

2.11. ネットワークのセキュリティー保護

ネットワークセキュリティーは、複数のレベルで管理できます。Pod レベルでは、ネットワーク namespace はネットワークアクセスを制限することで、コンテナーが他の Pod やホストシステムを認識できないようにすることができます。ネットワークポリシーを使用すると、接続の許可と拒否を制御できます。コンテナー化されたアプリケーションに対する ingress および egress トラフィックを管理することができます。

2.11.1. ネットワーク namespace の使用

OpenShift Container Platform はソフトウェア定義ネットワーク (SDN) を使用して、クラスター全体でのコンテナー間の通信を可能にする統一クラスターネットワークを提供します。

ネットワークポリシーモードは、デフォルトで他の Pod およびネットワークエンドポイントからプロジェクトのすべての Pod にアクセスできるようにします。プロジェクトで 1 つ以上の Pod を分離するには、そのプロジェクトで **NetworkPolicy** オブジェクトを作成し、許可する着信接続を指定します。マルチテナントモードを使用すると、Pod およびサービスのプロジェクトレベルの分離を実行できます。

2.11.2. ネットワークポリシーを使用した Pod の分離

ネットワークポリシー を使用して、同じプロジェクトの Pod を相互に分離することができます。ネットワークポリシーでは、Pod へのネットワークアクセスをすべて拒否し、Ingress コントローラーの接続のみを許可したり、他のプロジェクトの Pod からの接続を拒否したり、ネットワークの動作に関する同様のルールを設定したりできます。

関連情報

- ネットワークポリシーについて

2.11.3. 複数の Pod ネットワークの使用

実行中の各コンテナーには、デフォルトでネットワークインターフェイスが 1 つだけあります。Multus CNI プラグインを使用すると、複数の CNI ネットワークを作成し、それらのネットワークのいずれかを Pod に割り当てることができます。このようにして、プライベートデータをより制限されたネットワークに分離し、各ノードに複数のネットワークインターフェイスを持たせることができます。

関連情報

- 複数ネットワークの使用

2.11.4. アプリケーションの分離

OpenShift Container Platform では、ユーザー、チーム、アプリケーション、および環境を非グローバルリソースから分離するマルチテナントのクラスターを作成するために、単一のクラスター上でネットワークのトラフィックをセグメント化することができます。

関連情報

- OpenShiftSDN を使用したネットワーク分離の設定

2.11.5. Ingress トラフィックのセキュリティー保護

OpenShift Container Platform クラスター外から Kubernetes サービスへのアクセスを設定する方法に関連し、セキュリティー上の影響に関する多数の考慮点があります。Ingress ルーティングでは、HTTP および HTTPS ルートを公開するほか、NodePort または LoadBalancer Ingress タイプを設定できます。NodePort は、それぞれのクラスターワーカーからアプリケーションのサービス API オブジェクトを公開します。LoadBalancer を使用すると、外部ロードバランサーを OpenShift Container Platform クラスターの関連付けられたサービス API オブジェクトに割り当てることができます。

関連情報

- ingress クラスタートラフィックの設定

2.11.6. Egress トラフィックのセキュリティー保護

OpenShift Container Platform は、ルーターまたはファイアウォールのいずれかを使用して Egress トラフィックを制御する機能を提供します。たとえば、IP のホワイトリストを使用して、データベースのアクセスを制御できます。クラスター管理者は、1 つ以上の egress IP アドレスを OpenShift Container Platform SDN ネットワークプロバイダーのプロジェクトに割り当てることができます。同様に、クラスター管理者は egress ファイアウォールを使用して、egress トラフィックが OpenShift Container Platform クラスター外に送信されないようにできます。

固定 egress IP アドレスを割り当てることで、すべての送信トラフィックを特定プロジェクトのその IP アドレスに割り当てることができます。egress ファイアウォールを使用すると、Pod が外部ネットワークに接続されないようにしたり、Pod が内部ネットワークに接続されないようにするか、Pod の特定の内部サブネットへのアクセスを制限したりできます。

関連情報

- 外部 IP アドレスへのアクセスを制御するための egress ファイアウォールの設定
- プロジェクトの egress IP の設定

2.12. 割り当てられたストレージのセキュリティー保護

OpenShift Container Platform は、オンプレミスおよびクラウドプロバイダーの両方で、複数のタイプのストレージをサポートします。とくに、OpenShift Container Platform は Container Storage Interface をサポートするストレージタイプを使用できます。

2.12.1. 永続ボリュームプラグイン

コンテナーは、ステートレスとステートフルの両方のアプリケーションに役立ちます。割り当て済みのストレージを保護することは、ステートフルサービスのセキュリティーを保護する上で重要な要素になります。Container Storage Interface (CSI) を使用すると、OpenShift Container Platform は CSI インターフェイスをサポートするストレージバックエンドからのストレージを組み込むことができます。

OpenShift Container Platform は、以下を含む複数のタイプのストレージのプラグインを提供します。

- Red Hat OpenShift Data Foundation *
- AWS Elastic Block Stores (EBS) *
- AWS Elastic File System (EFS) *
- Azure Disk
- Azure File
- OpenStack Cinder *
- GCE Persistent Disks *
- VMware vSphere *
- ネットワークファイルシステム (NFS)
- FlexVolume
- ファイバーチャネル
- iSCSI

動的プロビジョニングでのこれらのストレージタイプのプラグインには、アスタリスク (*) が付いています。送信中のデータは、相互に通信している OpenShift Container Platform のすべてのコンポーネントに対して HTTPS 経由で暗号化されます。

永続ボリューム (PV) はストレージタイプでサポートされる方法でホスト上にマウントできます。異なるタイプのストレージにはそれぞれ異なる機能があり、各 PV のアクセスモードは、特定のボリュームによってサポートされる特定のモードに設定されます。

たとえば、NFS は複数の読み取り/書き込みクライアントをサポートしますが、特定の NFS PV は読み取り専用としてサーバー上でエクスポートされる可能性があります。各 PV には、**ReadWriteOnce**、**ReadOnlyMany**、および **ReadWriteMany** など、特定の PV 機能を説明したアクセスモードの独自のセットがあります。

2.12.2. 共有ストレージ

NFS のような共有ストレージプロバイダーの場合、PV はグループ ID (GID) を PV リソースのアノテーションとして登録します。次に、Pod が PV を要求する際に、アノテーションが付けられた GID が Pod の補助グループに追加され、この Pod に共有ストレージのコンテンツへのアクセスを付与します。

2.12.3. ブロックストレージ

AWS Elastic Block Store (EBS)、GCE Persistent Disks、および iSCSI などのブロックストレージプロバイダーの場合、OpenShift Container Platform は SELinux 機能を使用し、権限のない Pod のマウントされたボリュームについて、そのマウントされたボリュームが関連付けられたコンテナーにのみ所有され、このコンテナーにのみ表示されるようにしてそのルートを保護します。

関連情報

- 永続ストレージについて

- CSI ボリュームの設定
- 動的プロビジョニング
- NFS を使用した永続ストレージ
- AWS Elastic Block Store を使用した永続ストレージ
- GCE Persistent Disk を使用した永続ストレージ

2.13. クラスターイベントとログの監視

OpenShift Container Platform クラスターを監視および監査する機能は、不適切な利用に対してクラスターおよびそのユーザーを保護する上で重要な要素となります。

これに関連し、イベントとログという 2 つの主な情報源をクラスターレベルの情報として使用できます。

2.13.1. クラスターイベントの監視

クラスター管理者は、**Event** リソースタイプを理解し、システムイベントのリストを確認して、どのイベントが重要かを判断することを推奨します。イベントは、関連するリソースの namespace または **default** namespace (クラスターイベントの場合) のいずれかの namespace に関連付けられます。デフォルトの namespace は、クラスターを監視または監査するための関連するイベントを保持します。たとえば、これにはノードイベントおよびインフラストラクチャーコンポーネントに関連したリソースイベントが含まれます。

マスター API および **oc** コマンドは、イベントのリストをノードに関連するものに制限するパラメーターを提供しません。これを実行する簡単な方法として **grep** を使用することができます。

```
$ oc get event -n default | grep Node
```

出力例

```
1h      20h      3      origin-node-1.example.local   Node      Normal    NodeHasDiskPressure   ...
```

より柔軟な方法として、他のツールで処理できる形式でイベントを出力することができます。たとえば、次の例では、JSON 出力に対して **jq** ツールを使用して、**NodeHasDiskPressure** イベントのみを抽出します。

```
$ oc get events -n default -o json \
  | jq '.items[] | select(.involvedObject.kind == "Node" and .reason == "NodeHasDiskPressure")'
```

出力例

```
{
  "apiVersion": "v1",
  "count": 3,
  "involvedObject": {
    "kind": "Node",
    "name": "origin-node-1.example.local",
    "uid": "origin-node-1.example.local"
  },
```

```
  "kind": "Event",
  "reason": "NodeHasDiskPressure",
  ...
}
```

リソースの作成や変更、または削除に関連するイベントも、クラスターの不正な使用を検出するために使用することができます。たとえば、以下のクエリーは、イメージの過剰なプルの有無を確認するために使用できます。

```
$ oc get events --all-namespaces -o json \
  | jq '[.items[] | select(.involvedObject.kind == "Pod" and .reason == "Pulling")] | length'
```

出力例

```
4
```

> **注記**
>
> namespace を削除すると、そのイベントも削除されます。イベントも期限切れになる可能性があり、etcd ストレージが一杯にならないように削除されます。イベントは永続するレコードとして保存されず、一定期間の統計データを取得するためにポーリングを頻繁に実行する必要があります。

2.13.2. ロギング

oc log コマンドを使用して、コンテナーログ、ビルド設定およびデプロイメントをリアルタイムで表示できます。ユーザーによって、ログへの異なるアクセスが必要になる場合があります。

- プロジェクトにアクセスできるユーザーは、デフォルトでそのプロジェクトのログを確認することができます。
- 管理ロールを持つユーザーは、すべてのコンテナーログにアクセスできます。

詳細な監査および分析のためにログを保存するには、**cluster-logging** アドオン機能を有効にして、システム、コンテナー、監査ログを収集し、管理し、表示できます。OpenShift Elasticsearch Operator および Red Hat OpenShift Logging Operator を使用して OpenShift Logging をデプロイし、管理し、アップグレードできます。

2.13.3. 監査ログ

監査ログ を使用すると、ユーザー、管理者、またはその他の OpenShift Container Platform コンポーネントの動作に関連する一連のアクティビティーをフォローできます。API 監査ロギングは各サーバーで行われます。

関連情報

- システムイベントの一覧
- OpenShift Logging について
- 監査ログの表示

第3章 証明書の設定

3.1. デフォルトの INGRESS 証明書の置き換え

3.1.1. デフォルトの Ingress 証明書について

デフォルトで、OpenShift Container Platform は Ingress Operator を使用して内部 CA を作成し、**.apps** サブドメインの下にあるアプリケーションに有効なワイルドカード証明書を発行します。Web コンソールと CLI のどちらもこの証明書を使用します。

内部インフラストラクチャー CA 証明書は自己署名型です。一部のセキュリティーまたは PKI チームにとってこのプロセスは適切とみなされない可能性がありますが、ここで想定されるリスクは最小限のものです。これらの証明書を暗黙的に信頼するクライアントがクラスター内の他のコンポーネントになります。デフォルトのワイルドカード証明書を、コンテナーユーザー空間で提供される CA バンドルにすでに含まれているパブリック CA に置き換えることで、外部クライアントは **.apps** サブドメインで実行されるアプリケーションに安全に接続できます。

3.1.2. デフォルトの Ingress 証明書の置き換え

.apps サブドメインにあるすべてのアプリケーションのデフォルトの Ingress 証明書を置き換えることができます。証明書を置き換えると、Web コンソールや CLI を含むすべてのアプリケーションで、指定した証明書による暗号化が提供されます。

前提条件

- 完全修飾 **.apps** サブドメインおよびその対応するプライベートキーのワイルドカード証明書が必要です。それぞれが個別の PEM 形式のファイルである必要があります。
- プライベートキーの暗号化は解除されている必要があります。キーが暗号化されている場合は、これを OpenShift Container Platform にインポートする前に復号化します。
- 証明書には、***.apps.<clustername>.<domain>** を示す **subjectAltName** 拡張が含まれている必要があります。
- 証明書ファイルでは、チェーンに 1 つ以上の証明書を含めることができます。ファイルには、ワイルドカード証明書を最初の証明書としてリストし、その後に他の中間証明書をリストし、最後にルート CA 証明書をリストする必要があります。
- ルート CA 証明書を追加の PEM 形式のファイルにコピーします。
- **-----END CERTIFICATE-----** を含むすべての証明書が、その行の後に 1 つの改行で終了していることを確認します。

手順

1. ワイルドカード証明書の署名に使用されるルート CA 証明書のみを含む config map を作成します。

```
$ oc create configmap custom-ca \
     --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
     -n openshift-config
```

1. **</path/to/example-ca.crt>** は、ローカルファイルシステム上のルート CA 証明書ファイルへのパスです。たとえば、**/etc/pki/ca-trust/source/anchors** です。

2. 新たに作成された設定マップでクラスター全体のプロキシー設定を更新します。

```
$ oc patch proxy/cluster \
    --type=merge \
    --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

注記

クラスターの信頼された CA のみを更新する場合は、MCO によって **/etc/pki/ca-trust/source/anchors/openshift-config-user-ca-bundle.crt** ファイルが更新され、Machine Config Controller (MCC) によって信頼された CA の更新が各ノードに適用されます。そのため、ノードの再起動は不要です。ただし、これらの変更に伴い、Machine Config Daemon (MCD) によって各ノード上の重要なサービス (kubelet や CRI-O など) が再起動されます。これらのサービスの再起動により、サービスが完全に再起動されるまで、各ノードは一時的に **NotReady** 状態になります。

openshift-config-user-ca-bundle.crt ファイル内の他のパラメーター (**noproxy** など) を変更すると、MCO によってクラスター内の各ノードが再起動されます。

3. ワイルドカード証明書チェーンおよびキーが含まれるシークレットを作成します。

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-ingress
```

1. **<secret>** は、証明書チェーンおよびプライベートキーが含まれるシークレットの名前です。
2. **</path/to/cert.crt>** は、ローカルファイルシステム上の証明書チェーンへのパスです。
3. **</path/to/cert.key>** は、この証明書に関連付けられるプライベートキーへのパスです。

4. Ingress コントローラー設定を、新規に作成されたシークレットで更新します。

```
$ oc patch ingresscontroller.operator default \
    --type=merge -p \
    '{"spec":{"defaultCertificate": {"name": "<secret>"}}}' \ 1
    -n openshift-ingress-operator
```

1. **<secret>** を、直前の手順でシークレットに使用された名前に置き換えます。

重要

ローリング更新を実行するために Ingress Operator をトリガーするには、シークレットの名前を更新する必要があります。kubelet はボリュームマウント内のシークレットへの変更を自動的に伝播するため、シークレットの内容を更新してもローリング更新はトリガーされません。詳細は、この Red Hat Knowledgebase Solution を参照してください。

3.1.3. 関連情報

- CA バンドル証明書の置き換え
- プロキシー証明書のカスタマイズ

3.2. API サーバー証明書の追加

デフォルトの API サーバー証明書は、内部 OpenShift Container Platform クラスター CA によって発行されます。クラスター外のクライアントは、デフォルトで API サーバーの証明書を検証できません。この証明書は、クライアントが信頼する CA によって発行される証明書に置き換えることができます。

注記

Hosted Control Plane クラスターでは、API から自己署名証明書を置き換えることはできません。

3.2.1. API サーバーの名前付き証明書の追加

デフォルトの API サーバー証明書は、内部 OpenShift Container Platform クラスター CA によって発行されます。リバースプロキシーやロードバランサーが使用される場合など、クライアントが要求する完全修飾ドメイン名 (FQDN) に基づいて、API サーバーが返す代替証明書を 1 つ以上追加できます。

前提条件

- FQDN とそれに対応するプライベートキーの証明書が必要です。それぞれが個別の PEM 形式のファイルである必要があります。
- プライベートキーの暗号化は解除されている必要があります。キーが暗号化されている場合は、これを OpenShift Container Platform にインポートする前に復号化します。
- 証明書には、FQDN を示す **subjectAltName** 拡張が含まれる必要があります。
- 証明書ファイルでは、チェーンに 1 つ以上の証明書を含めることができます。API サーバー FQDN の証明書は、ファイルの最初の証明書である必要があります。この後には中間証明書が続き、ファイルの最後はルート CA 証明書にすることができます。

警告

内部ロードバランサーに名前付きの証明書を指定しないようにしてください (ホスト名 **api-int.<cluster_name>.<base_domain>**)。これを指定すると、クラスターの状態は動作の低下した状態になります。

手順

1. **kubeadmin** ユーザーとして新しい API にログインします。

```
$ oc login -u kubeadmin -p <password> https://FQDN:6443
```

2. **kubeconfig** ファイルを取得します。

```
$ oc config view --flatten > kubeconfig-newapi
```

3. **openshift-config** namespace に証明書およびプライベートキーが含まれるシークレットを作成します。

```
$ oc create secret tls <secret> \ 1
    --cert=</path/to/cert.crt> \ 2
    --key=</path/to/cert.key> \ 3
    -n openshift-config
```

1 **<secret>** は、証明書チェーンおよびプライベートキーが含まれるシークレットの名前です。

2 **</path/to/cert.crt>** は、ローカルファイルシステム上の証明書チェーンへのパスです。

3 **</path/to/cert.key>** は、この証明書に関連付けられるプライベートキーへのパスです。

4. API サーバーを作成されたシークレットを参照するように更新します。

```
$ oc patch apiserver cluster \
    --type=merge -p \
    '{"spec":{"servingCerts": {"namedCertificates": \
    [{"names": ["<FQDN>"], \ 1
    "servingCertificate": {"name": "<secret>"}}]}}}' 2
```

1 **<FQDN>** を、API サーバーが証明書を提供する FQDN に置き換えます。ポート番号を含めないでください。

2 **<secret>** を、直前の手順でシークレットに使用された名前に置き換えます。

5. **apiserver/cluster** オブジェクトを検査し、シークレットが参照されていることを確認します。

```
$ oc get apiserver cluster -o yaml
```

出力例

```
...
spec:
  servingCerts:
    namedCertificates:
    - names:
      - <FQDN>
```

```
    servingCertificate:
      name: <secret>
...
```

6. **kube-apiserver** Operator を確認し、Kubernetes API サーバーの新しいリビジョンがロールアウトされることを確認します。Operator が設定の変更を検出して新しいデプロイメントをトリガーするのに 1 分かかる場合があります。新しいリビジョンが公開されている間、**PROGRESSING** は **True** を報告します。

```
$ oc get clusteroperators kube-apiserver
```

以下の出力にあるように、**PROGRESSING** が **False** と表示されるまで次の手順に移行しないでください。

出力例

```
NAME             VERSION   AVAILABLE   PROGRESSING   DEGRADED   SINCE
kube-apiserver   4.16.0    True        False         False      145m
```

PROGRESSING が **True** と表示されている場合は、数分待機してから再試行します。

注記

Kubernetes API サーバーの新しいリビジョンは、API サーバーの名前付き証明書が初めて追加された場合にのみロールアウトされます。API サーバーの名前付き証明書が更新されると、**kube-apiserver** Pod が更新された証明書を動的に再ロードするため、Kubernetes API サーバーの新しいリビジョンはロールアウトされません。

3.3. サービス提供証明書のシークレットによるサービストラフィックのセキュリティー保護

3.3.1. サービス提供証明書について

サービス提供証明書は、暗号化を必要とする複雑なミドルウェアアプリケーションをサポートすることが意図されています。これらの証明書は、TLS Web サーバー証明書として発行されます。

service-ca コントローラーは、サービス証明書を生成するために **x509.SHA256WithRSA** 署名アルゴリズムを使用します。

生成される証明書およびキーは PEM 形式のもので、作成されたシークレット内の **tls.crt** および **tls.key** にそれぞれ保存されます。証明書およびキーは、有効期間に近づくと自動的に置き換えられます。

サービス証明書を発行するサービス CA 証明書は 26 ヵ月間有効であり、有効期間が 13 ヵ月未満になると自動的にローテーションされます。ローテーション後も、直前のサービス CA 設定は有効期限が切れるまで信頼されます。これにより、影響を受けるすべてのサービスについて、期限が切れる前にそれらのキーの情報を更新できるように猶予期間が許可されます。この猶予期間中にクラスターをアップグレード (サービスを再起動してそれらのキー情報を更新する) を実行しない場合、直前のサービス CA の期限が切れた後の失敗を防ぐためにサービスを手動で再起動する必要がある場合があります。

注記

以下のコマンドを使用して、クラスター内のすべての Pod を手動で再起動できます。このコマンドは、すべての namespace で実行されているすべての Pod を削除するため、このコマンドを実行するとサービスが中断します。これらの Pod は削除後に自動的に再起動します。

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
    do oc delete pods --all -n $I; \
    sleep 1; \
    done
```

3.3.2. サービス証明書の追加

サービスとの通信のセキュリティーを保護するには、サービスと同じ namespace のシークレットに署名済みの提供証明書とキーのペアを生成します。

生成される証明書は、内部サービス DNS 名 **<service.name>.<service.namespace>.svc** にのみ有効であり、内部通信用にのみ有効です。サービスがヘッドレスサービス (**clusterIP** 値が設定されていない) である場合、生成された証明書には ***.<service.name>.<service.namespace>.svc** 形式のワイルドカードのサブジェクトも含まれます。

重要

生成された証明書にはヘッドレスサービスのワイルドカードサブジェクトが含まれるため、クライアントが個別の Pod を区別する必要がある場合はサービス CA を使用しないでください。この場合は、以下のようになります。

- 別の CA を使用して個別の TLS 証明書を生成します。
- サービス CA は、個々の Pod に送信される接続に関する信頼される CA として許可することはできず、他の Pod がこの権限を借用することはできません。これらの接続は、個別の TLS 証明書の生成に使用されている CA を信頼するように設定される必要があります。

前提条件

- サービスが定義されていること。

手順

1. サービスに **service.beta.openshift.io/serving-cert-secret-name** のアノテーションを付けます。

```
$ oc annotate service <service_name> \ (1)
    service.beta.openshift.io/serving-cert-secret-name=<secret_name> (2)
```

(1) **<service_name>** を、セキュリティー保護するサービスの名前に置き換えます。

(2) **<secret_name>** は、証明書とキーのペアを含む生成されたシークレットの名前です。

注記

この値は、利便性のために、**<service_name>** と同じにすることを推奨します。

たとえば、以下のコマンドを使用してサービス **test1** にアノテーションを付けます。

```
$ oc annotate service test1 service.beta.openshift.io/serving-cert-secret-name=test1
```

2. アノテーションが存在することを確認するためにサービスを検査します。

```
$ oc describe service <service_name>
```

出力例

```
...
Annotations:              service.beta.openshift.io/serving-cert-secret-name: <service_name>
                          service.beta.openshift.io/serving-cert-signed-by: openshift-service-serving-signer@1556850837
...
```

3. クラスターがサービスのシークレットを生成した後に、**Pod** 仕様がこれをマウントでき、Pod はシークレットが利用可能になった後にこれを実行できます。

関連情報

- サービス証明書を使用して、reencrypt TLS 終端を使用してセキュアなルートを設定できます。詳細は、カスタム証明書を使用した re-encrypt ルートの作成 を参照してください。

3.3.3. サービス CA バンドルの設定マップへの追加

Pod は、**service.beta.openshift.io/inject-cabundle=true** アノテーションを持つ **ConfigMap** オブジェクトをマウントすることで、サービスの認証局 (CA) 証明書にアクセスできます。config map にアノテーションが付けられると、クラスターがサービスの CA 証明書を config map の **service-ca.crt** キーに自動的に注入します。この CA 証明書へのアクセスにより、TLS クライアントがサービス提供証明書を使用してサービスへの接続を検証することが可能になります。

重要

OpenShift Service CA Operator は、このアノテーションを config map に追加した後、config map 内のすべてのデータを削除します。Pod の設定を格納している config map と同じものを使用するのではなく、別の config map を使用して **service-ca.crt** を格納することを検討してください。

手順

1. 次のコマンドを入力して、config map に **service.beta.openshift.io/inject-cabundle=true** アノテーションを付けます。

```
$ oc annotate configmap <config_map_name> \
    service.beta.openshift.io/inject-cabundle=true
```

1. **<config_map_name>** を、アノテーションを付ける設定マップの名前に置き換えます。

注記

ボリュームマウントで **service-ca.crt** キーを明示的に参照すると、config map に CA バンドルが注入されるまで Pod が起動しなくなります。この動作は、ボリュームのサービス証明書設定で **optional** パラメーターを **true** に設定することでオーバーライドできます。

2. config map を表示して、サービス CA バンドルが注入されていることを確認します。

```
$ oc get configmap <config_map_name> -o yaml
```

CA バンドルは、YAML 出力の **service-ca.crt** キーの値として表示されます。

```
apiVersion: v1
data:
  service-ca.crt: |
    -----BEGIN CERTIFICATE-----
...
```

3. **Deployment** オブジェクトを設定して、Pod 内に存在する各コンテナーに config map をボリュームとしてマウントします。

マウントされる config map のボリュームを定義する Deployment オブジェクトの例

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: my-example-custom-ca-deployment
  namespace: my-example-custom-ca-ns
spec:
  ...
    spec:
      ...
      containers:
        - name: my-container-that-needs-custom-ca
          volumeMounts:
          - name: trusted-ca
            mountPath: /etc/pki/ca-trust/extracted/pem
            readOnly: true
      volumes:
      - name: trusted-ca
        configMap:
          name: <config_map_name> 1
          items:
            - key: ca-bundle.crt 2
              path: tls-ca-bundle.pem 3
# ...
```

1. この手順の前のステップでアノテーションを付けた config map の名前を指定します。

2. **ca-bundle.crt** は ConfigMap キーとして必要になります。

3. **tls-ca-bundle.pem** は ConfigMap パスとして必要になります。

3.3.4. サービス CA バンドルの API サービスへの追加

APIService オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、その **spec.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。

手順

1. API サービスに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate apiservice <api_service_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1. **<api_service_name>** を、アノテーションを付ける API サービスの名前に置き換えます。

たとえば、以下のコマンドを使用して API サービス **test1** にアノテーションを付けます。

```
$ oc annotate apiservice test1 service.beta.openshift.io/inject-cabundle=true
```

2. API サービスを表示し、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get apiservice <api_service_name> -o yaml
```

CA バンドルは YAML 出力の **spec.caBundle** フィールドに表示されます。

```
apiVersion: apiregistration.k8s.io/v1
kind: APIService
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  caBundle: <CA_BUNDLE>
...
```

3.3.5. サービス CA バンドルのカスタムリソース定義への追加

CustomResourceDefinition (CRD) オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、その **spec.conversion.webhook.clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。

注記

サービス CA バンドルは、CRD が変換に Webhook を使用するように設定されている場合にのみ CRD にインジェクトされます。CRD の Webhook がサービス CA 証明書でセキュリティー保護されている場合にのみ、サービス CA バンドルを挿入することは役に立ちます。

手順

1. CRD に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate crd <crd_name> \ (1)
    service.beta.openshift.io/inject-cabundle=true
```

(1) **<crd_name>** をアノテーションを付ける CRD の名前に置き換えます。

たとえば、以下のコマンドを使用して CRD **test1** にアノテーションを付けます。

```
$ oc annotate crd test1 service.beta.openshift.io/inject-cabundle=true
```

2. CRD を表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get crd <crd_name> -o yaml
```

CA バンドルは、YAML 出力の **spec.conversion.webhook.clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: apiextensions.k8s.io/v1
kind: CustomResourceDefinition
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
spec:
  conversion:
    strategy: Webhook
    webhook:
      clientConfig:
        caBundle: <CA_BUNDLE>
...
```

3.3.6. サービス CA バンドルの変更用 Webhook 設定への追加

MutatingWebhookConfiguration オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、各 Webhook の **clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。

注記

異なる Webhook に異なる CA バンドルを指定する必要がある受付 Webhook 設定にはこのアノテーションを設定しないでください。これを実行する場合、サービス CA バンドルはすべての Webhook に挿入されます。

手順

1. 変更用 Webhook 設定に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate mutatingwebhookconfigurations <mutating_webhook_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<mutating_webhook_name>** を、アノテーションを付ける変更用 Webhook 設定の名前に置き換えます。

たとえば、以下のコマンドを使用して変更用 webhook 設定 **test1** にアノテーションを付けます。

```
$ oc annotate mutatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. 変更用 webhook 設定を表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get mutatingwebhookconfigurations <mutating_webhook_name> -o yaml
```

CA バンドルは、YAML 出力のすべての Webhook の **clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: admissionregistration.k8s.io/v1
kind: MutatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.7. サービス CA バンドルの変更用 webhook 設定への追加

ValidatingWebhookConfiguration オブジェクトに **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付け、各 Webhook の **clientConfig.caBundle** フィールドにサービス CA バンドルを設定できます。これにより、Kubernetes API サーバーはターゲットに設定されたエンドポイントのセキュリティーを保護するために使用されるサービス CA 証明書を検証することができます。

> **注記**
>
> 異なる Webhook に異なる CA バンドルを指定する必要がある受付 Webhook 設定にはこのアノテーションを設定しないでください。これを実行する場合、サービス CA バンドルはすべての Webhook に挿入されます。

手順

1. 検証用 Webhook 設定に **service.beta.openshift.io/inject-cabundle=true** のアノテーションを付けます。

```
$ oc annotate validatingwebhookconfigurations <validating_webhook_name> \ 1
    service.beta.openshift.io/inject-cabundle=true
```

1 **<validating_webhook_name>** をアノテーションを付ける検証用 Webhook 設定の名前に置き換えます。

たとえば、以下のコマンドを使用して検証用 webhook 設定 **test1** にアノテーションを付けます。

```
$ oc annotate validatingwebhookconfigurations test1 service.beta.openshift.io/inject-cabundle=true
```

2. 検証用 webhook 設定を表示して、サービス CA バンドルが挿入されていることを確認します。

```
$ oc get validatingwebhookconfigurations <validating_webhook_name> -o yaml
```

CA バンドルは、YAML 出力のすべての Webhook の **clientConfig.caBundle** フィールドに表示されます。

```
apiVersion: admissionregistration.k8s.io/v1
kind: ValidatingWebhookConfiguration
metadata:
  annotations:
    service.beta.openshift.io/inject-cabundle: "true"
...
webhooks:
- myWebhook:
  - v1beta1
  clientConfig:
    caBundle: <CA_BUNDLE>
...
```

3.3.8. 生成されたサービス証明書の手動によるローテーション

関連付けられたシークレットを削除することにより、サービス証明書をローテーションできます。シークレットを削除すると、新規のシークレットが自動的に作成され、新規証明書が作成されます。

前提条件

- 証明書とキーのペアを含むシークレットがサービス用に生成されていること。

手順

1. 証明書を含むシークレットを確認するためにサービスを検査します。これは、以下に示すように **serving-cert-secret-name** アノテーションにあります。

```
$ oc describe service <service_name>
```

出力例

```
...
service.beta.openshift.io/serving-cert-secret-name: <secret>
...
```

2. サービスの生成されたシークレットを削除します。このプロセスで、シークレットが自動的に再作成されます。

```
$ oc delete secret <secret> 1
```

1 **<secret>** を、直前の手順のシークレットの名前に置き換えます。

3. 新規シークレットを取得し、**AGE** を調べて、証明書が再作成されていることを確認します。

```
$ oc get secret <service_name>
```

出力例

```
NAME            TYPE              DATA   AGE
<service.name>    kubernetes.io/tls   2      1s
```

3.3.9. サービス CA 証明書の手動によるローテーション

サービス CA は 26 ヵ月間有効で、有効期間が 13 ヵ月未満になると自動的に更新されます。

必要に応じて、以下の手順でサービス CA を手動で更新することができます。

> **警告**
>
> 手動でローテーションされるサービス CA は、直前のサービス CA で信頼を維持しません。クラスターの Pod が再起動するまでサービスが一時的に中断する可能性があります。これにより、Pod が新規サービス CA で発行されるサービス提供証明書を使用できるようになります。

前提条件

- クラスター管理者としてログインしている必要があります。

手順

1. 以下のコマンドを使用して、現在のサービス CA 証明書の有効期限を表示します。

```
$ oc get secrets/signing-key -n openshift-service-ca \
     -o template='{{index .data "tls.crt"}}' \
     | base64 --decode \
     | openssl x509 -noout -enddate
```

2. サービス CA を手動でローテーションします。このプロセスは、新規サービス証明書に署名するために使用される新規サービス CA を生成します。

```
$ oc delete secret/signing-key -n openshift-service-ca
```

3. 新規証明書をすべてのサービスに適用するには、クラスター内のすべての Pod を再起動します。このコマンドにより、すべてのサービスが更新された証明書を使用するようになります。

```
$ for I in $(oc get ns -o jsonpath='{range .items[*]} {.metadata.name}{"\n"} {end}'); \
      do oc delete pods --all -n $I; \
      sleep 1; \
      done
```

警告

このコマンドは、すべての namespace で実行されているすべての Pod を調べ、これらを削除するため、サービスを中断させます。これらの Pod は削除後に自動的に再起動します。

3.4. CA バンドルの更新

3.4.1. CA バンドル証明書について

プロキシー証明書により、ユーザーは egress 接続の実行時にプラットフォームコンポーネントによって使用される 1 つ以上のカスタム認証局 (CA) を指定できます。

プロキシーオブジェクトの **trustedCA** フィールドは、ユーザーによって提供される信頼される認証局 (CA) バンドルを含む設定マップの参照です。このバンドルは Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージされ、egress HTTPS 呼び出しを行うプラットフォームコンポーネントの信頼ストアに挿入されます。たとえば、**image-registry-operator** は外部イメージレジストリーを呼び出してイメージをダウンロードします。**trustedCA** が指定されていない場合、RHCOS 信頼バンドルのみがプロキシーされる HTTPS 接続に使用されます。独自の証明書インフラストラクチャーを使用する場合は、カスタム CA 証明書を RHCOS 信頼バンドルに指定します。

trustedCA フィールドは、プロキシーバリデーターによってのみ使用される必要があります。バリデーターは、必要なキー **ca-bundle.crt** から証明書バンドルを読み取り、これを **openshift-config-managed** namespace の **trusted-ca-bundle** という名前の設定マップにコピーします。**trustedCA** によって参照される設定マップの namespace は **openshift-config** です。

```
apiVersion: v1
kind: ConfigMap
```

```
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

3.4.2. CA バンドル証明書の置き換え

手順

1. ワイルドカード証明書の署名に使用されるルート CA 証明書が含まれる設定マップを作成します。

```
$ oc create configmap custom-ca \
     --from-file=ca-bundle.crt=</path/to/example-ca.crt> \ 1
     -n openshift-config
```

1 **</path/to/example-ca.crt>** は、ローカルファイルシステム上の CA 証明書バンドルへのパスです。

2. 新たに作成された設定マップでクラスター全体のプロキシー設定を更新します。

```
$ oc patch proxy/cluster \
     --type=merge \
     --patch='{"spec":{"trustedCA":{"name":"custom-ca"}}}'
```

3.4.3. 関連情報

- デフォルトの Ingress 証明書の置き換え
- クラスター全体のプロキシーの有効化
- プロキシー証明書のカスタマイズ

第4章 証明書の種類および説明

4.1. API サーバーのユーザーによって提供される証明書

4.1.1. 目的

API サーバーは、**api.<cluster_name>.<base_domain>** のクラスター外にあるクライアントからアクセスできます。クライアントに別のホスト名で API サーバーにアクセスさせたり、クラスター管理の認証局 (CA) 証明書をクライアントに配布せずに API サーバーにアクセスさせたりする必要が生じる場合があります。管理者は、コンテンツを提供する際に API サーバーによって使用されるカスタムデフォルト証明書を設定する必要があります。

4.1.2. 場所

ユーザーによって提供される証明書は、**openshift-config** namespace の **kubernetes.io/tls** タイプの **Secret** で指定される必要があります。ユーザーによって提供される証明書を使用できるように、API サーバークラスター設定の **apiserver/cluster** リソースを更新します。

4.1.3. 管理

ユーザーによって提供される証明書はユーザーによって管理されます。

4.1.4. 有効期限

API サーバークライアント証明書の有効期限は 5 分未満です。

ユーザーによって提供される証明書はユーザーによって管理されます。

4.1.5. カスタマイズ

必要に応じて、ユーザーが管理する証明書を含むシークレットを更新します。

4.1.6. 関連情報

- API サーバー証明書の追加

4.2. プロキシー証明書

4.2.1. 目的

プロキシー証明書により、ユーザーは egress 接続の実行時にプラットフォームコンポーネントによって使用される 1 つ以上のカスタム認証局 (CA) 証明書を指定できます。

プロキシーオブジェクトの **trustedCA** フィールドは、ユーザーによって提供される信頼される認証局 (CA) バンドルを含む設定マップの参照です。このバンドルは Red Hat Enterprise Linux CoreOS (RHCOS) 信頼バンドルにマージされ、egress HTTPS 呼び出しを行うプラットフォームコンポーネントの信頼ストアに挿入されます。たとえば、**image-registry-operator** は外部イメージレジストリーを呼び出してイメージをダウンロードします。**trustedCA** が指定されていない場合、RHCOS 信頼バンドルのみがプロキシーされる HTTPS 接続に使用されます。独自の証明書インフラストラクチャーを使用する場合は、カスタム CA 証明書を RHCOS 信頼バンドルに指定します。

trustedCA フィールドは、プロキシーバリデーターによってのみ使用される必要があります。バリデーターは、必要なキー **ca-bundle.crt** から証明書バンドルを読み取り、これを **openshift-config-managed** namespace の **trusted-ca-bundle** という名前の設定マップにコピーします。**trustedCA** によって参照される設定マップの namespace は **openshift-config** です。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: user-ca-bundle
  namespace: openshift-config
data:
  ca-bundle.crt: |
    -----BEGIN CERTIFICATE-----
    Custom CA certificate bundle.
    -----END CERTIFICATE-----
```

4.2.1.1. 関連情報

- クラスター全体のプロキシーの設定

4.2.2. インストール時のプロキシー証明書の管理

インストーラー設定の **additionalTrustBundle** 値は、インストール時にプロキシー信頼 CA 証明書を指定するために使用されます。以下に例を示します。

```
$ cat install-config.yaml
```

出力例

```
...
proxy:
  httpProxy: http://<username:password@proxy.example.com:123/>
  httpsProxy: http://<username:password@proxy.example.com:123/>
  noProxy: <123.example.com,10.88.0.0/16>
additionalTrustBundle: |
    -----BEGIN CERTIFICATE-----
   <MY_HTTPS_PROXY_TRUSTED_CA_CERT>
    -----END CERTIFICATE-----
...
```

4.2.3. 場所

ユーザーによって提供される信頼バンドルは、設定マップとして表現されます。設定マップは、egress HTTPS 呼び出しを行うプラットフォームコンポーネントのファイルシステムにマウントされます。通常、Operator は設定マップを **/etc/pki/ca-trust/extracted/pem/tls-ca-bundle.pem** にマウントしますが、これはプロキシーでは必要ありません。プロキシーは HTTPS 接続を変更したり、検査したりできます。いずれの場合も、プロキシーは接続用の新規証明書を生成して、これに署名する必要があります。

完全なプロキシーサポートとは、指定されたプロキシーに接続し、生成した署名を信頼することを指します。そのため、信頼されたルートに接続しているいずれの証明書チェーンも信頼されるように、ユーザーがその信頼されたルートを指定する必要があります。

RHCOS トラストバンドルを使用する場合は、CA 証明書を **/etc/pki/ca-trust/source/anchors** に配置します。詳細は、Red Hat Enterprise Linux (RHEL) の **ネットワークのセキュリティー保護** ドキュメントの 共有システム証明書の使用 を参照してください。

4.2.4. 有効期限

ユーザーは、ユーザーによって提供される信頼バンドルの有効期限を設定します。

デフォルトの有効期限は CA 証明書自体で定義されます。この設定は、OpenShift Container Platform または RHCOS で使用する前に、CA 管理者が証明書に対して行います。

注記

Red Hat では、CA の有効期限が切れるタイミングを監視しません。ただし、CA の有効期間は長く設定されるため、通常問題は生じません。ただし、信頼バンドルを定期的に更新する必要がある場合があります。

4.2.5. サービス

デフォルトで、egress HTTPS 呼び出しを行うすべてのプラットフォームコンポーネントは RHCOS 信頼バンドルを使用します。**trustedCA** が定義される場合、これも使用されます。

RHCOS ノードで実行されているすべてのサービスは、ノードの信頼バンドルを使用できます。

4.2.6. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.2.7. カスタマイズ

ユーザーによって提供される信頼バンドルを更新するには、以下のいずれかを実行します。

- **trustedCA** で参照される設定マップの PEM でエンコードされた証明書の更新
- 新しい信頼バンドルが含まれる namespace **openshift-config** での設定マップの作成、および新規設定マップの名前を参照できるようにするための **trustedCA** の更新

CA 証明書を RHCOS 信頼バンドルに書き込むメカニズムは、machine config を使用して行われるその他のファイルの RHCOS への書き込みと全く同じです。Machine Config Operator (MCO) は、新しい CA 証明書を含む新しい machine config を適用した後、**update-ca-trust** プログラムを実行し、RHCOS ノード上で CRI-O サービスを再起動します。この更新に、ノードの再起動は必要ありません。CRI-O サービスを再起動すると、新しい CA 証明書で信頼バンドルが自動的に更新されます。以下に例を示します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
    machineconfiguration.openshift.io/role: worker
  name: 50-examplecorp-ca-cert
spec:
  config:
    ignition:
      version: 3.1.0
```

```
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-
8;base64,LS0tLS1CRUdJTiBDRVJUSUZJQ0FURS0tLS0tCk1JSUVORENDQXh5Z0F3SUJBZ0lKQU5
1bkkwRDY2MmNuTUEwR0NTcUdTSWIzRFFFQkN3VUFNSUdsTVFzd0NRWUQKV1FRR0V3SlZVek
VYTUJVR0ExVUVDQXdPVG05eWRHZ2dRMkZ5YjJ4cGJtRXhFREFPQmdOVkJBY01CMUpoYkdWcA
pBMmd4RmpBVUJnTlZCQW9NRFZKbFpDQklZWFFzSUVsdVl5NHhFekFSQmdOVkJBc01DbEpsWk
NCSVlYUWdTVlF4Ckh6QVpCZ05WQkFNTUVsSmxaQ0JJWVhRZ1NWUWdVbTl2ZENCRFFURWhN
QjhHQ1NxR1NJYjNEUUVKQVJZU2FXNW0KWGpDQnBURUxNQWtHQTFVRUJoTUNWVk14RnpBV
kJnTlZCQWdNRGs1dmNuUm9JRU5oY205c2FXNWhNUkF3RGdZRApXUVFIREFkU1lXeGxhV2RvTV
JZd0ZBWURWUVFLREExU1pXUWdTR0YwTENCSmJtTXVNUk13RVFZRFZRUUxEQXBTCkFXUWd
TR0YwSUVsVU1Sc3dHUVlEVlFRRERCSlNaV1FnU0dGMElFbFVJRkp2YjNRZ1EwRXhJVEFmQmdrc
WhraUcKMHcwQkNRRVdFbWx1Wm05elpXTkFjbVZrYUdGMExtTnZiVENDQVNJd0RRWUpLb1pJaH
ZjTkFRRUJCUUFFZ2dFUApCRENDQVFvQ2dnRUJBTFF0OU9KUWg2R0M1TFQxZzgwcU5oMHU1
MEJRNHNaL3laOGFFVHh0KzVsblBWWDZNSEt6CmQvaTdsRHFUZlRjZkxMMm55VUJkMmZRRGsx
QjBmeHJza2hHSUlaM2lmUDFQczRsdFRrdjhoUlNvYjNWdE5xU28KSHhrS2Z2RDJQS2pUUHhEUFdZ
eXJ1eTlpckxxaW9NZmZpM2kvZ0N1dDBaV3RBeU8zTVZJNXFXRi9lbkt3Z1BFUwpZOXBvK1RkQ3ZS
Qi9SVU9iQmFNNzYxRWNyTFNNMUdxSE51ZVNmcW5obzNBakxRNmRCblBXbG82MzhabTFWZWJ
LCkNFTHloa0xXTVNGa0t3RG1uZTBqUTAyWTRnMDc1dkNLdkNzQ0F3RUFBYU5qTUdFd0hRWUR
WUjBPQkJZRUZJN1lKNXlDK1VlaElJUGV1TDhacXczUHpiZ2NaTUI4R0ExVWRJd1FZTUJhQUZJN1I0
eUMrVWVoSUlQZXVMOFpxdzNQegpjZ2NaTUE4R0ExVWRFd0VCL3dRRk1BTUJBZjh3RGdZRFZS
MFBBUUgvQkFRREFnR0dNQTBHQ1NxR1NJYjNEUUVCCkR3VUFBNElCQVFCRE52RDJWbTlzQT
VBOUFsT0pSOCtlbjVYejloWGN4SkI1cGh4Y1pROGpGb0cwNFZzaHZkMGUKTUVuVXJNY2ZGZ0laN
G5qTUtUUUNNNFpGVVBBaWV5THg0ZjUySHVEb3BwM2U1SnlJTWZXK0tGY05JcEt3Q3NhawpU2
9LdElVT3NVSks3cUJWWnhjckl5ZVFWMnFjWU9lWmh0UzV3QnFJd09BaEZ3bENFVDdaZTU4UUhtUz
Q4c2xqCjVlVGtSaml2QWxFeHJGektjbGpDNGF4S1Fsbk92VkF6eitHbTMyVTB4UEJGNEJ5ZVBWeEN
KVUh3MVRzeVRtZWwKU3hORXA3eUhvWGN3bitmWG5hK3Q1SldoMWd4VVp0eTMKLS0tLS1FTkQ
gQ0VSVElGSUNBVEUtLS0tLQo=
        mode: 0644
        overwrite: true
        path: /etc/pki/ca-trust/source/anchors/examplecorp-ca.crt
```

マシンの信頼ストアは、ノードの信頼ストアの更新もサポートする必要があります。

4.2.8. 更新

RHCOS ノードで証明書を自動更新できる Operator はありません。

注記

Red Hat では、CA の有効期限が切れるタイミングを監視しません。ただし、CA の有効期間は長く設定されるため、通常問題は生じません。ただし、信頼バンドルを定期的に更新する必要がある場合があります。

4.3. サービス CA 証明書

4.3.1. 目的

service-ca は、OpenShift Container Platform クラスターのデプロイ時に自己署名の CA を作成する Operator です。

4.3.2. 有効期限

カスタムの有効期限はサポートされません。自己署名 CA は、フィールド **tls.crt** (証明書)、**tls.key** (プライベートキー)、および **ca-bundle.crt** (CA バンドル) の修飾名 **service-ca/signing-key** を持つシークレットに保存されます。

他のサービスは、サービスリソースに **service.beta.openshift.io/serving-cert-secret-name: <secret name>** のアノテーションを付けてサービス提供証明書を要求できます。応答として、Operator は、名前付きシークレットに対し、新規証明書を **tls.crt** として、プライベートキーを **tls.key** として生成します。証明書は 2 年間有効です。

他のサービスは、サービス CA から生成される証明書の検証をサポートするために、**service.beta.openshift.io/inject-cabundle: true** のアノテーションを付けてサービス CA の CA バンドルを API サービスまたは設定マップリソースに挿入するように要求します。応答として、Operator はその現在の CA バンドルを API サービスの **CABundle** フィールドに書き込むか、**service-ca.crt** として設定マップに書き込みます。

OpenShift Container Platform 4.3.5 の時点で、自動ローテーションはサポートされ、一部の 4.2.z および 4.3.z リリースにバックポートされます。自動ローテーションをサポートするすべてのリリースに対して、サービス CA は 26 ヵ月間有効であり、有効期間までの残りの期間が 13 ヵ月未満になると自動的に更新されます。必要に応じて、サービス CA を手動で更新することができます。

サービス CA 有効期限の 26 ヵ月は、サポートされる OpenShift Container Platform クラスターの予想されるアップグレード間隔よりも長くなります。そのため、サービス CA 証明書のコントロールプレーン以外のコンシューマーは CA のローテーション後に更新され、またローテーション前の CA の有効期限が切れる前に更新されます。

警告

手動でローテーションされるサービス CA は、直前のサービス CA で信頼を維持しません。クラスターの Pod が再起動するまでサービスが一時的に中断する可能性があります。これにより、Pod が新規サービス CA で発行されるサービス提供証明書を使用できるようになります。

4.3.3. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.3.4. サービス

サービス CA 証明書を使用するサービスには以下が含まれます。

- cluster-autoscaler-operator
- cluster-monitoring-operator
- cluster-authentication-operator
- cluster-image-registry-operator
- cluster-ingress-operator
- cluster-kube-apiserver-operator

- cluster-kube-controller-manager-operator
- cluster-kube-scheduler-operator
- cluster-networking-operator
- cluster-openshift-apiserver-operator
- cluster-openshift-controller-manager-operator
- cluster-samples-operator
- cluster-storage-operator
- machine-config-operator
- console-operator
- insights-operator
- machine-api-operator
- operator-lifecycle-manager
- CSI ドライバー Operator

これはすべてを網羅したリストではありません。

4.3.5. 関連情報

- サービス提供証明書の手動ローテーション
- サービス提供証明書のシークレットによるサービストラフィックのセキュリティー保護

4.4. ノード証明書

4.4.1. 目的

ノード証明書はクラスターによって署名され、kubelet が Kubernetes API サーバーと通信できるようにします。これらは、ブートストラッププロセスで生成される kubelet CA 証明書から取得されます。

4.4.2. 場所

kubelet CA 証明書は、**openshift-kube-apiserver-operator** namespace の **kube-apiserver-to-kubelet-signer** シークレットにあります。

4.4.3. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.4.4. 有効期限

ノード証明書は 30 日後に自動的にローテーションされます。

4.4.5. 更新

Kubernetes API Server Operator は、292 日後に新規の **kube-apiserver-to-kubelet-signer** CA 証明書を自動的に生成します。古い CA 証明書は 365 日後に削除されます。kubelet CA 証明書を更新または削除しても、ノードは再起動されません。

クラスター管理者は、次のコマンドを実行して kubelet CA 証明書を手動で更新できます。

```
$ oc annotate -n openshift-kube-apiserver-operator secret kube-apiserver-to-kubelet-signer
auth.openshift.io/certificate-not-after-
```

4.4.6. 関連情報

- ノードの使用

4.5. ブートストラップ証明書

4.5.1. 目的

OpenShift Container Platform 4 以降では、kubelet は **/etc/kubernetes/kubeconfig** にあるブートストラップ証明書を使用して初回のブートストラップを実行します。その次に、ブートストラップの初期化プロセス および CSR を作成するための kubelet の認証 に進みます。

このプロセスでは、kubelet はブートストラップチャネル上での通信中に CSR を生成します。コントローラーマネージャーは CSR に署名すると、kubelet が管理する証明書が作成されます。

4.5.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.5.3. 有効期限

このブートストラップ証明書は 10 年間有効です。

kubelet が管理する証明書は 1 年間有効であり、その 1 年の約 80 パーセントマークで自動的にローテーションします。

> **注記**
>
> OpenShift Lifecycle Manager (OLM) は、ブートストラップ証明書を更新しません。

4.5.4. カスタマイズ

ブートストラップ証明書をカスタマイズすることはできません。

4.6. ETCD 証明書

4.6.1. 目的

etcd 証明書は etcd-signer によって署名されます。それらの証明書はブートストラッププロセスで生成される認証局 (CA) から提供されます。

4.6.2. 有効期限

CA 証明書は 10 年間有効です。ピア、クライアント、およびサーバーの証明書は 3 年間有効です。

4.6.3. etcd 証明書のローテーション

有効期限が切れる前に **etcd** 証明書をローテーションします。

手順

1. 次のコマンドを実行して、現在の署名者証明書のバックアップコピーを作成します。

```
$ oc get secret -n openshift-etcd etcd-signer -oyaml > signer_backup_secret.yaml
```

2. 次のコマンドを実行して、新しい署名者証明書の残りの有効期間を確認します。

```
$ oc get secret -n openshift-etcd etcd-signer -
ojsonpath='{.metadata.annotations.auth\.openshift\.io/certificate-not-after}'
```

3. 残りの有効期間が現在の日付に近い場合は、署名者を削除して再作成し、静的 Pod のロールアウトを待ちます。

   - 次のコマンドを実行して署名者を削除します。

   ```
   $ oc delete secret -n openshift-etcd etcd-signer
   ```

   - 次のコマンドを実行して、静的 Pod のロールアウトを待ちます。

   ```
   $ oc wait --for=condition=Progressing=False --timeout=15m clusteroperator/etcd
   ```

4. **etcd** が再起動したら、次のコマンドを実行して、**openshift-config** namespace の元の CA を、**openshift-etcd** 内の新たにローテーションされた CA に切り替えます。

```
$ oc get secret etcd-signer -n openshift-etcd -ojson | jq
'del(.metadata["namespace","creationTimestamp","resourceVersion","selfLink","uid"])' | oc
apply -n openshift-config -f -
```

5. 次のコマンドを実行して、クラスター Operator がロールアウトされ、安定するまで待ちます。

```
$ oc adm wait-for-stable-cluster --minimum-stable-period 2m
```

4.6.4. etcd 証明書ローテーションアラートおよびメトリクス署名者証明書

2 種類のアラートにより、保留中の **etcd** 証明書の有効期限がユーザーに通知されます。

etcdSignerCAExpirationWarning

署名者の有効期限が切れるまで 730 日間通知されます。

etcdSignerCAExpirationCritical

署名者の有効期限が切れるまで 365 日間通知されます。

証明書をローテーションできる理由は次のとおりです。

- 有効期限切れのアラートが表示された場合
- 秘密鍵が漏洩した場合

重要

秘密鍵が漏洩した場合は、すべての証明書をローテーションする必要があります。

OpenShift Container Platform のメトリクスシステムには、専用の **etcd** 署名者が存在します。**etcd 証明書のローテーション** では、次のメトリクス用のパラメーターに置き換えてください。

- **etcd-signer** の代わりに **etcd-metric-signer** を使用
- **etcd-ca-bundle** の代わりに **etcd-metrics-ca-bundle** を使用

4.6.5. 管理

これらの証明書はシステムによってのみ管理され、自動的にローテーションされます。

4.6.6. サービス

etcd 証明書は、etcd メンバーピアと暗号化されたクライアントトラフィック間の暗号化された通信に使用されます。以下の証明書は etcd および etcd と通信する他のプロセスによって生成され、使用されます。

- ピア証明書: etcd メンバー間の通信に使用されます。
- クライアント証明書: 暗号化されたサーバーとクライアント間の通信に使用されます。現時点で、クライアント証明書は API サーバーによってのみ使用され、プロキシーを除いてその他のサービスは etcd に直接接続されません。クライアントシークレット (**etcd-client**、**etcd-metric-client**、**etcd-metric-signer**、**etcd-signer**) が、**openshift-config**、**openshift-etcd**、**openshift-etcd-operator**、および **openshift-kube-apiserver** namespace に追加されます。
- サーバー証明書: クライアント要求を認証するために etcd サーバーによって使用されます。
- メトリック証明書: メトリックのすべてのコンシューマーは metric-client 証明書を使用してプロキシーに接続します。

4.6.7. 関連情報

- クラスターの直前の状態への復元

4.7. OLM 証明書

4.7.1. 管理

Operator Lifecycle Manager (OLM) コンポーネント (**olm-operator**、**catalog-operator**、**packageserver**、および **marketplace-operator**) のすべての証明書はシステムによって管理されます。

Webhook または API サービスを含む Operator を **ClusterServiceVersion** (CSV) オブジェクトにインストールする場合、OLM はこれらのリソースの証明書を作成し、ローテーションします。**openshift-operator-lifecycle-manager** namespace のリソースの証明書は OLM によって管理されます。

OLM はプロキシー環境で管理する Operator の証明書を更新しません。これらの証明書は、ユーザーがサブスクリプション設定で管理する必要があります。

次のステップ

- Operator Lifecycle Manager でのプロキシーサポートの設定

4.7.2. 関連情報

- プロキシー証明書
- デフォルトの Ingress 証明書の置き換え
- CA バンドルの更新

4.8. 集合 API クライアント証明書

4.8.1. 目的

集約 API クライアント証明書は、集約 API サーバーに接続するときに KubeAPIServer を認証するために使用されます。

4.8.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.8.3. 有効期限

この CA は 30 日間有効です。

管理クライアント証明書は 30 日間有効です。

CA およびクライアント証明書は、コントローラーを使用して自動的にローテーションされます。

4.8.4. カスタマイズ

集約された API サーバー証明書をカスタマイズすることはできません。

4.9. MACHINE CONFIG OPERATOR 証明書

4.9.1. 目的

この認証局は、初期プロビジョニング中にノードから Machine Config Server (MCS) への接続を保護するために使用されます。

証明書は 2 つあります。

- 自己署名 CA である MCS CA

- 派生証明書、MCS 証明書

4.9.1.1. プロビジョニングの詳細

Red Hat Enterprise Linux CoreOS (RHCOS) を使用する OpenShift Container Platform インストールは、Ignition を使用してインストールされます。このプロセスは 2 つの部分に分かれています。

1. MCS によって提供される完全な設定の URL を参照する Ignition 設定が作成されます。
2. ユーザーがプロビジョニングしたインフラストラクチャーのインストール方法の場合、Ignition 設定は、**openshift-install** コマンドによって作成された **worker.ign** ファイルとしてマニフェストされます。Machine API Operator を使用する、インストーラーがプロビジョニングするインフラストラクチャーのインストール方法の場合、この設定は **worker-user-data** シークレットとして表示されます。

重要

現在、マシン設定サーバーエンドポイントをブロックまたは制限する方法はサポートされていません。マシン設定サーバーは、既存の設定または状態を持たない新しくプロビジョニングされたマシンが設定を取得できるように、ネットワークに公開する必要があります。このモデルでは、信頼のルートは証明書署名要求 (CSR) エンドポイントであり、kubelet がクラスターに参加するための承認のために証明書署名要求を送信する場所です。このため、シークレットや証明書などの機密情報を配布するためにマシン設定を使用しないでください。

マシン設定サーバーエンドポイント、ポート 22623 および 22624 がベアメタルシナリオで確実に保護されるようにするには、顧客は適切なネットワークポリシーを設定する必要があります。

関連情報

- Machine Config Operator
- OpenShift SDN ネットワークプラグインについて

4.9.1.2. 信頼チェーンのプロビジョニング

MCS CA は、**security.tls.certificateAuthorities** 設定フィールドの下の Ignition 設定に挿入されます。その後、MCS は、Web サーバーによって提示された MCS 証明書を使用して完全な設定を提供します。

クライアントは、サーバーによって提示された MCS 証明書に、認識する機関への信頼チェーンがあることを検証します。この場合、MCS CA がその認証局であり、MCS 証明書に署名します。これにより、クライアントが正しいサーバーにアクセスしていることを確認できます。この場合のクライアントは、initramfs のマシン上で実行されている Ignition です。

4.9.1.3. クラスター内のキーマテリアル

MCS CA は、**ca.crt** キーを持つ **kube-system** namespace、**root-ca** オブジェクト内の config map としてクラスターに表示されます。秘密鍵はクラスターに保存されず、インストール完了後に破棄されます。

MCS 証明書は、**tls.crt** キーと **tls.key** キーを持つ **openshift-machine-config-operator** namespace および **machine-config-server-tls** オブジェクト内のシークレットとしてクラスターに表示されます。

4.9.2. 管理

現時点では、これらの証明書のいずれかを直接変更することはサポートされていません。

4.9.3. 有効期限

MCS CA は 10 年間有効です。

発行されたサービング証明書は 10 年間有効です。

4.9.4. カスタマイズ

Machine Config Operator 証明書をカスタマイズすることはできません。

4.10. デフォルト INGRESS のユーザーによって提供される証明書

4.10.1. 目的

アプリケーションは通常 **<route_name>.apps.<cluster_name>.<base_domain>** で公開されます。**<cluster_name>** および **<base_domain>** はインストール設定ファイルから取得されます。**<route_name>** は、(指定されている場合) ルートのホストフィールド、またはルート名です。例: **hello-openshift-default.apps.username.devcluster.openshift.com**.**hello-openshift** はルートの名前で、ルートは default namespace に置かれます。クラスター管理の CA 証明書をクライアントに分散せずに、クライアントにアプリケーションにアクセスさせる必要がある場合があります。管理者は、アプリケーションコンテンツを提供する際にカスタムのデフォルト証明書を設定する必要があります。

> **警告**
>
> Ingress Operator は、カスタムのデフォルト証明書を設定するまで、プレースホルダーとして機能する Ingress コントローラーのデフォルト証明書を生成します。実稼働クラスターで Operator が生成するデフォルト証明書を使用しないでください。

4.10.2. 場所

ユーザーによって提供される証明書は、**openshift-ingress** namespace の **tls** タイプの **Secret** で指定される必要があります。ユーザーがユーザーによって提供される証明書を有効にできるようにするために、**openshift-ingress-operator** namespace で **IngressController** CR を更新します。このプロセスの詳細は、カスタムデフォルト証明書の設定 を参照してください。

4.10.3. 管理

ユーザーによって提供される証明書はユーザーによって管理されます。

4.10.4. 有効期限

ユーザーによって提供される証明書はユーザーによって管理されます。

4.10.5. サービス

クラスターにデプロイされるアプリケーションは、デフォルト Ingress にユーザーによって提供される証明書を使用します。

4.10.6. カスタマイズ

必要に応じて、ユーザーが管理する証明書を含むシークレットを更新します。

4.10.7. 関連情報

- デフォルトの Ingress 証明書の置き換え

4.11. INGRESS 証明書

4.11.1. 目的

Ingress Operator は以下の目的で証明書を使用します。

- Prometheus のメトリックへのアクセスのセキュリティーを保護する。
- ルートへのアクセスのセキュリティーを保護する。

4.11.2. 場所

Ingress Operator および Ingress コントローラーメトリックへのアクセスのセキュリティーを保護するために、Ingress Operator はサービス提供証明書を使用します。Operator は独自のメトリックに対して **service-ca** コントローラーから証明書を要求し、**service-ca** コントローラーは証明書を **openshift-ingress-operator** namespace の **metrics-tls** という名前のシークレットに配置します。さらに、Ingress Operator は各 Ingress コントローラーの証明書を要求し、**service-ca** コントローラーは証明書を **router-metrics-certs-<name>** という名前のシークレットに配置します。ここで、**<name>** は **openshift-ingress** namespace の Ingress コントローラーの名前です。

各 Ingress コントローラーには、独自の証明書を指定しないセキュリティー保護されたルートに使用するデフォルト証明書があります。カスタム証明書を指定しない場合、Operator はデフォルトで自己署名証明書を使用します。Operator は独自の自己署名証明書を使用して、生成するデフォルト証明書に署名します。Operator はこの署名証明書を生成し、これを **openshift-ingress-operator** namespace の **router-ca** という名前のシークレットに配置します。Operator がデフォルトの証明書を生成する際に、デフォルト証明書を **openshift-ingress** namespace の **router-certs-<name>** という名前のシークレットに配置します (ここで、**<name>** は Ingress コントローラーの名前です)。

> **警告**
>
> Ingress Operator は、カスタムのデフォルト証明書を設定するまで、プレースホルダーとして機能する Ingress コントローラーのデフォルト証明書を生成します。実稼働クラスターで Operator が生成するデフォルト証明書は使用しないでください。

4.11.3. ワークフロー

図4.1 カスタム証明書のワークフロー

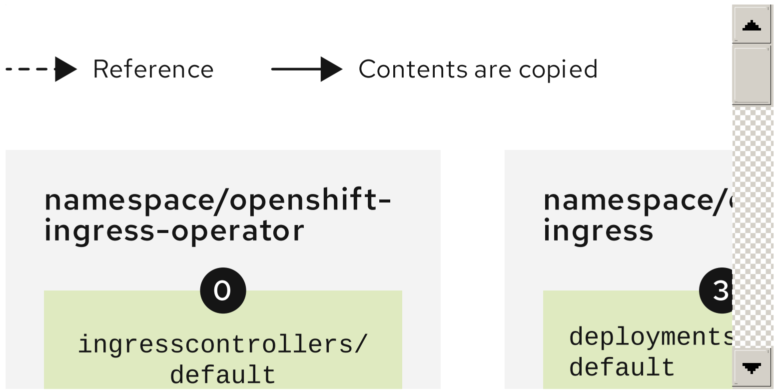
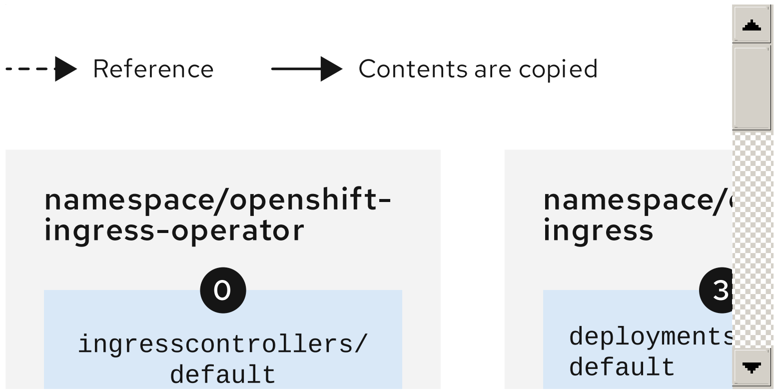


図4.2 デフォルトの証明書ワークフロー



0 空の **defaultCertificate** フィールドにより、Ingress Operator はその自己署名 CA を使用して指定されたドメインの提供証明書を生成します。

1 Ingress Operator によって生成されるデフォルトの CA 証明書およびキー。Operator が生成するデフォルトの提供証明書に署名するために使用されます。

2 デフォルトのワークフローでは、Ingress Operator によって作成され、生成されるデフォルト CA 証明書を使用して署名されるワイルドカードのデフォルト提供証明書です。カスタムワークフローでは、これはユーザーによって提供される証明書です。

3 ルーターのデプロイメント。**secrets/router-certs-default** の証明書を、デフォルトのフロントエンドサーバー証明書として使用します。

4 デフォルトのワークフローでは、ワイルドカードのデフォルト提供証明書 (パブリックおよびプライベートの部分) の内容がここにコピーされ、OAuth 統合が有効になります。カスタムワークフローでは、これはユーザーによって提供される証明書です。

5 デフォルト提供証明書のパブリック (証明書) の部分です。**configmaps/router-ca** リソースを置き換えます。

6 ユーザーは **ingresscontroller** 提供証明書に署名した CA 証明書でクラスタープロキシー設定を更新します。これにより、**auth**、**console** などのコンポーネントや、提供証明書を信頼するために使用するレジストリーが有効になります。

7 ユーザーバンドルが指定されていない場合に、組み合わせた Red Hat Enterprise Linux CoreOS (RHCOS) およびユーザーによって提供される CA バンドルまたは RHCOS のみのバンドルを含むクラスター全体の信頼される CA バンドルです。

8 他のコンポーネント (**auth** および **console** など) がカスタム証明書で設定された **ingresscontroller** を信頼するよう指示するカスタム CA 証明書バンドルです。

9 **trustedCA** フィールドは、ユーザーによって提供される CA バンドルを参照するように使用されます。

10 Cluster Network Operator は、信頼される CA バンドルを **proxy-ca** 設定マップに挿入します。

11 OpenShift Container Platform 4.16 以降は **default-ingress-cert** を使用します。

4.11.4. 有効期限

Ingress Operator の証明書の有効期限は以下の通りです。

- **service-ca** コントローラーが作成するメトリック証明書の有効期限は、作成日から 2 年間です。
- Operator の署名証明書の有効期限は、作成日から 2 年間です。
- Operator が生成するデフォルト証明書の有効期限は、作成日から 2 年間です。

Ingress Operator または **service-ca** コントローラーが作成する証明書のカスタム有効期限を指定することはできません。

Ingress Operator または **service-ca** コントローラーが作成する証明書に OpenShift Container Platform をインストールする場合に、有効期限を指定することはできません。

4.11.5. サービス

Prometheus はメトリックのセキュリティーを保護する証明書を使用します。

Ingress Operator はその署名証明書を使用して、カスタムのデフォルト証明書を設定しない Ingress コントローラー用に生成するデフォルト証明書に署名します。

セキュリティー保護されたルートを使用するクラスターコンポーネントは、デフォルトの Ingress コントローラーのデフォルト証明書を使用できます。

セキュリティー保護されたルート経由でのクラスターへの Ingress は、ルートが独自の証明書を指定しない限り、ルートがアクセスされる Ingress コントローラーのデフォルト証明書を使用します。

4.11.6. 管理

Ingress 証明書はユーザーによって管理されます。詳細は、デフォルト ingress 証明書の置き換え を参照してください。

4.11.7. 更新

service-ca コントローラーは、これが発行する証明書を自動的にローテーションします。ただし、**oc delete secret <secret>** を使用してサービス提供証明書を手動でローテーションすることができます。

Ingress Operator は、独自の署名証明書または生成するデフォルト証明書をローテーションしません。Operator が生成するデフォルト証明書は、設定するカスタムデフォルト証明書のプレースホルダーとして使用されます。

4.12. モニタリングおよび OPENSHIFT LOGGING OPERATOR コンポーネント証明書

4.12.1. 有効期限

モニタリングコンポーネントは、サービス CA 証明書でトラフィックのセキュリティーを保護します。これらの証明書は 2 年間有効であり、13 ヵ月ごとに実行されるサービス CA のローテーションで自動的に置き換えられます。

証明書が **openshift-monitoring** または **openshift-logging** namespace にある場合、これはシステムで管理され、自動的にローテーションされます。

4.12.2. 管理

これらの証明書は、ユーザーではなく、システムによって管理されます。

4.13. コントロールプレーンの証明書

4.13.1. 場所

コントロールプレーンの証明書はこれらの namespace に含まれます。

- openshift-config-managed
- openshift-kube-apiserver
- openshift-kube-apiserver-operator
- openshift-kube-controller-manager
- openshift-kube-controller-manager-operator
- openshift-kube-scheduler

4.13.2. 管理

コントロールプレーンの証明書はシステムによって管理され、自動的にローテーションされます。

稀なケースとしてコントロールプレーンの証明書の有効期限が切れる場合は、コントロールプレーン証明書の期限切れの状態からのリカバリー を参照してください。

第5章 COMPLIANCE OPERATOR

5.1. COMPLIANCE OPERATOR の概要

OpenShift Container Platform Compliance Operator は、多数の技術実装の検査を自動化することでユーザーを支援し、それらを業界標準、ベンチマーク、およびベースラインの特定の要素と比較します。ただし、Compliance Operator は監査人ではありません。このようなさまざまな標準に対する準拠または認定を実現するには、Qualified Security Assessor (QSA)、Joint Authorization Board (JAB)、または業界で認められたその他の規制当局など、認定監査機関に依頼して、環境の評価を受ける必要があります。

Compliance Operator は、このような標準に関する一般に入手可能な情報とプラクティスに基づいて推奨事項を作成し、場合によっては修復を支援します。ただし、実際の準拠はお客様の責任となります。標準への準拠を実現するには、認定監査人と協力する必要があります。最新の更新については、Compliance Operator リリースノート を参照してください。すべての Red Hat 製品のコンプライアンスサポートの詳細は、Product Compliance を参照してください。

5.1.1. Compliance Operator の概念

Compliance Operator について

カスタムリソース定義を理解する

5.1.2. Compliance Operator の管理

Compliance Operator のインストール

Compliance Operator の更新

Compliance Operator の管理

Compliance Operator のアンインストール

5.1.3. Compliance Operator のスキャンの管理

サポートされているコンプライアンスプロファイル

Compliance Operator のスキャン

Compliance Operator の調整

Compliance Operator の未加工の結果の取得

コンプライアンス Operator 修復の管理

高度な Compliance Operator タスクの実行

Compliance Operator のトラブルシューティング

oc-compliance プラグインの使用

5.2. COMPLIANCE OPERATOR リリースノート

Compliance Operator を使用すると、OpenShift Container Platform 管理者はクラスターの必要なコンプライアンス状態を記述し、存在するギャップやそれらを修復する方法に関する概要を提供します。

以下のリリースノートは、OpenShift Container Platform の Compliance Operator の開発履歴を記録したものです。

Compliance Operator の概要は、Understanding the Compliance Operator を参照してください。

最新リリースにアクセスするには、コンプライアンス Operator の更新 を参照してください。

すべての Red Hat 製品のコンプライアンスサポートの詳細は、Product Compliance を参照してください。

5.2.1. OpenShiftCompliance Operator1.8.0

OpenShift Compliance Operator 1.8.0 に関する次のアドバイザリーが利用可能です。

- RHSA-2025:21885 - OpenShift Compliance Operator 1.8.0 のバグ修正と機能強化更新

5.2.1.1. 新機能および機能拡張

- この更新により、Compliance Operator は TECH PREVIEW ステータスの Common Expression Language (CEL) スキャナーを提供します。CEL スキャナーは、管理者が CEL 式を使用してカスタムセキュリティーポリシーを定義および適用できるようにする新しい **CustomRule** カスタムリソース定義 (CRD) を実装します。この新しいコンテンツ形式は、既存の XCCDF (Extensible Configuration Checklist Description Format) プロファイルを置き換えるものではなく、カスタムセキュリティーポリシーに準拠する機能を拡張します。詳細は (CMP-3118) を参照してください。

- 以前は、Compliance Operator では生のスキャン結果を保存するために永続的なストレージが必要であり、エッジデプロイメントやストレージインフラストラクチャーのない環境では課題となっていました。このリリースでは、Compliance Operator は永続ストレージなしでのスキャンの実行をサポートします。管理者は、**ScanSetting** リソースで **rawResultStorage.enabled: false** を設定してスキャン結果ファイルの保存を無効にし、エッジデプロイメントやシングルノードの OpenShift などのストレージが制限された環境でコンプライアンススキャンを実行できるようにすることができます。コンプライアンスチェックの結果は、**ComplianceCheckResult** リソースを通じて完全に利用できます。下位互換性を保つため、生の結果の保存はデフォルトで有効になっています。詳細は (CMP-1225) を参照してください。

- 以前は、Compliance Operator は、BSI コンプライアンススキャン用に **ocp4-bsi** および **ocp4-bsi-node** プロファイルを提供していました。このリリースでは、**rhcos4-bsi** プロファイルが利用可能になり、BSI 標準の範囲が RHCOS システムに拡張されました。詳細は (CMP-3720) を参照してください。

- このリリースでは、廃止された CIS 1.4.0、CIS 1.5.0、DISA STIG V1R1、および DISA STIG V2R1 プロファイルが削除されます。新しいバージョンでは、これらの古いプロファイルがお客様用に置き換えられました。詳細は (CMP-3712) を参照してください。

- このリリースでは、PCI-DSS プロファイル 3.2.1 および 4.0.0 が ARM アーキテクチャーシステムでサポートされるようになりました。詳細は、(CMP-3723) を参照してください。

5.2.1.2. バグ修正

- このリリースでは、API サーバー暗号化の自動修復により、OpenShift のバージョンに基づいて適切な暗号化モードが適用されるようになりました。OpenShift 4.13.0 以降のバージョンで

は AES-GCM、それより前のバージョンでは AES-CBC です。両方の暗号化モードは、すべての OpenShift バージョン間で準拠し続けます。詳細は (CMP-3248) を参照してください。

- このリリースより前では、Compliance Operator はすべての SSH 強化設定を含む固定の sshd_config ファイルをデプロイすることで、RHCOS ホスト上の SSH 設定を修正していました。対応するルールのスキャンが失敗した場合、SSH に意図しない設定変更が生じる可能性があります。このリリースでは、Compliance Operator は、https://github.com/ComplianceAsCode/content/blob/master/shared/macros/10-kubernetes.jinja#L1-L154 に示されているルールに従って、SSH に非常に具体的な修復を適用します。詳細は (CMP-3553) を参照してください。

- 以前のバージョンの Compliance Operator では、ログローテーション機能は **/etc/cron.daily** フォルダー内の **logrotate** ファイルの検索に依存していました。このリリースでは、Compliance Operator は **logrotate.timer** サービスと連携します。これにより、Compliance Operator からの信頼性の高いログローテーション動作が提供されます。

- 以前のバージョンの Compliance Operator では、コンプライアンスレポートから **STIG ID** を省略できる可能性があります。これらの省略は、**stigref 値** と **stigid** 値の欠落によって発生しました。このリリースでは、欠落が修正され、コンプライアンスレポートに **STIG ID が** 確実に表示されるようになりました。

- このリリースより前では、Compliance OperatorSTIG コントロール CNTR-OS-000720 はルール **rhcos4-audit-rules-suid- 特権 -function** を選択しましたが、このルールは Compliance Operator で使用できなかったため、出力は生成されませんでした。このリリースでは、ルール **rhcos4-audit-rules-suid- 特権 -function が** Compliance Operator で使用できるようになり、スキャン出力にリストされるようになりました。詳細は、(CMP-3558) を参照してください。

- 以前のバージョンの Compliance Operator では、TLS が正しく有効になっている場合でも、**ocp4-stig** プロファイルを使用したスキャンは、ルール **ocp4-stig-modified-audit-log-forwarding-uses-tls** で失敗していました。これは、**ClusterLogForwarder** リソースで **tls://** フィールドが不要になり、スキャン出力に誤った **FAIL** 結果が表示されるために発生します。このリリースでは、プロトコル接頭辞は必要なくなり、スキャン出力で正しい結果が生成されます。詳細は、ODF 4.11 がインストールされている場合に routes-protected-by-tls コンプライアンスチェックが失敗する を参照してください。

- 以前は、CIS ベンチマークコントロール 1.2.31 または 1.2.33 で推奨されているように、API サーバーがサポートされていない設定オーバーライドを使用しているかどうかを自動的に確認する方法はありませんでした。このリリースでは、サポートされていない設定のオーバーライドをチェックするための専用ルールが提供されます。

- Compliance Operator の以前のリリースでは、ルール **resource-requests-limits** など、一部のルールのアノテーションに変数参照がありませんでした。このリリースでは、変数参照がルールで使用できるようになり、誤った出力が排除されました。詳細は、(CMP-3582) を参照してください。

- 以前は、**ocp4-routes-rate-limit** ルールでは、**openshift** および **kube** 名前空間の外部にあるすべてのルートにレート制限を設定する必要がありました。ただし、重要な Operator によって管理される他の名前空間は変更されるべきではなく、Compliance Operator によって変更がスキャンされるべきではないため、この機能を使用してスキャンすると問題が発生します。このリリースでは、重要な Operator によって管理されるルートは、Compliance Operator によってエラーとしてフラグ付けされません。

- 以前のバージョンの Compliance Operator では、**openshift-sdn** ネットワークプロバイダーが見つからない場合、**ComplianceScan** によって **SDN が見つかりません** という警告が報告されていました。このリリースでは、OpenShift-SDN がアクティブなネットワークプロバイダーで

はない場合に、Compliance Operator によって警告が抑制されます。詳細は (CMP-3591) を参照してください。

- 以前は、**TailoredProfile** で重複した変数が誤って作成され、Compliance Operator によって正しく検出されない可能性がありました。このリリースでは、**TailoredProfile** 内の重複した **setValues** が識別され、コンプライアンススキャンから警告イベントがトリガーされます。

- Compliance Operator の以前のリリースでは、**clusterlogforwarder** 出力設定に URL キーのないマップが含まれている場合、ルール ocp4-audit-log-forwarding-uses-tls が失敗していました。このリリースでは、ルールは URL フィールドを持つ出力を正しくフィルタリングし、**clusterlogforwarder** に対して TLS が適切に有効になっている場合に **PASS** を表示します。詳細は、(CMP-3597) を参照してください。

- 以前のバージョンの Compliance Operator では、ルール **rhcos4-service-systemd-coredump-disabled** の場合、クラスターのスキャン後に修復は生成されませんでした。このリリースでは、**rhcos4-service-systemd-coredump-disabled** の修復が提供されています。

- 以前のバージョンの Compliance Operator では、**imagestream.spec.tags.importPolicy.scheduled** の設定をチェックするルールは、設定が正しい場合でも **FAIL** を返していました。このリリースでは、ルールによって、サンプルオペレーターによって管理されるイメージストリームと ClusterVersion によって所有されるイメージストリームが正しく除外されるようになり、コンプライアンスステータスのレポートが正確に行われるようになりました。

- 以前のリリースでは、Compliance Operator には、サポートされていない設定オーバーライドと不具合のある修復機能を使用する古い TLS 暗号スイートルールが含まれていました。このリリースでは、これらの古いルールはデフォルトのプロファイルから削除されました。また、整理しやすくするために、**ocp4-kubelet-configure-tls-cipher-suites-ingresscontroller** ルールの名前が **ocp4-ingress-controller-tls-cipher-suites** に変更されました。詳細は、(CMP-3606) を参照してください。

- 以前のバージョンの Compliance Operator では、カスタムコンテンツイメージを使用して **ComplianceScans を** 直接作成すると、プロファイルの非推奨チェック中に失敗しました。このリリースでは、Compliance Operator は **ProfileBundle を** 判別できない場合を適切に処理し、スキャンを失敗させる代わりに情報メッセージをログに記録します。詳細は、(CMP-3613) を参照してください。

- 以前は、Compliance Operator は **、ocp4-routes-protected-by-tls** ルールに準拠していないと誤ってフラグ付けされたパススルールートをスキャンしていました。このリリースでは、パススルールートは TLS 終了をバックエンドアプリケーションに委任するため、このルールから適切に除外されます。

5.2.2. OpenShift Compliance Operator 1.7.1

OpenShift Compliance Operator 1.7.1 については、次のアドバイザリーが利用できます。

- RHBA-2025:14639 - OpenShift Compliance Operator 1.7.1 bug fix update

5.2.2.1. バグ修正

- 以前は、Compliance Operator の **一時停止** コンテナーはメモリー不足により終了し、ステータスが表示されることがありました。 **OOMKilled)。この更新により、`pauser`のメモリー制限が**コンテナーを増やすことでエラーを防ぎ、全体的な安定性が向上します。(OCPBUGS-50924)

5.2.3. OpenShift Compliance Operator 1.7.0

OpenShift Compliance Operator 1.7.0 については、次のアドバイザリーが利用できます。

- RHBA-2025:3728 - OpenShift Compliance Operator 1.7.0 のバグ修正と機能拡張の更新

5.2.3.1. 新機能および機能拡張

- **aarch64**、**x86**、**ppc64le**、および **s390x** アーキテクチャーにインストールされた Compliance Operator で、**must-gather** 拡張機能が利用できるようになりました。**must-gather** ツールは、Red Hat Customer Support チームおよびエンジニアリングチームに重要な設定の詳細を提供するものです。詳細は、Compliance Operator の must-gather ツールの使用 を参照してください。

- CIS Benchmark のサポートが Compliance Operator 1.7.0 に追加されました。サポートされているプロファイルは、CIS OpenShift Benchmark 1.7.0 です。詳細は、(CMP-3081) を参照してください。

- Compliance Operator が、CIS OpenShift Benchmark 1.7.0 および FedRAMP Moderate Revision 4 の **aarch64** アーキテクチャーでサポートされるようになりました。詳細は、(CMP-2960) を参照してください。

- Compliance Operator 1.7.0 で、OpenShift および RHCOS の OpenShift DISA STIG V2R2 プロファイルがサポートされるようになりました。詳細は、(CMP-3142) を参照してください。

- Compliance Operator 1.7.0 で、サポートされていない古いプロファイルバージョンの非推奨 (CIS 1.4 プロファイル、CIS 1.5 プロファイル、DISA STIG V1R1 プロファイル、DISA STIG V2R1 プロファイルの非推奨など) がサポートされるようになりました。詳細は、(CMP-3149) を参照してください。

- Compliance Operator 1.7.0 のリリースにより、古い CIS および DISA STIG プロファイルが非推奨になりました。これらの古いプロファイルは、Compliance Operator 1.8.0 の公開とともにサポートされなくなる予定です。詳細は、(CMP-3284) を参照してください。

- Compliance Operator 1.7.0 のリリースにより、OpenShift に BSI プロファイルのサポートが追加されました。詳細は、KCS の記事 BSI クイックチェック および **BSI コンプライアンスサマリー** を参照してください。

5.2.3.2. バグ修正

- このリリースの前は、Compliance Operator が不要な修復推奨事項を提供していました。これは、**s390x** アーキテクチャーのファイルシステム構造の違いが原因でした。このリリースでは、Compliance Operator がファイルシステム構造の違いを認識するようになり、誤解を招く修復を提供しなくなりました。この更新により、ルールがより明確に定義されるようになりました。(OCPBUGS-33194)

- 以前は、ルール **ocp4-etcd-unique-ca** の指示が OpenShift 4.17 以降で機能しませんでした。この更新により、この指示と実行可能な手順が修正されました。(OCPBUGS-42350)

- Compliance Operator を Cluster Logging Operator (CLO) バージョン 6.0 と併用すると、さまざまなルールが失敗します。これは、CLO が使用する CRD に対する下位互換性のない変更が原因です。Compliance Operator は、この CRD を利用してロギング機能を検証しています。この CRD が修正され、PCI-DSS プロファイルと CLO の併用がサポートされるようになりました。(OCPBUGS-43229)

- ユーザーが Cluster Logging Operator (CLO) 6.0 をインストールすると、

ComplianceCheckResult の **ocp4-cis-audit-log-forwarding-enabled** が失敗していました。これは、**clusterlogforwarder** リソースの APIversion の変更が原因でした。現在は、ログの収集と転送の設定が新しい API で指定されるようになりました。この API は、observability.openshift.io API グループに含まれています。(OCPBUGS-43585)

- Compliance Operator の以前のリリースでは、スキャンによって Operator Pod のリコンサイルループのエラーログが生成されていました。このリリースでは、Compliance Operator コントローラーのロジックの安定性が向上しました。(OCPBUGS-51267)
- 以前は、**file-integrity-exists** または **file-integrity-notification-enabled** ルールが、**aarch64** OpenShift クラスターで失敗していました。この更新により、これらのルールが **aarch64** システムでは **NOT-APPLICABLE** と評価されるようになりました。(OCPBUGS-52884)
- この Compliance Operator のリリース前は、ルール **kubelet-configure-tls-cipher-suites** が API サーバーの暗号で失敗し、**E2E-FAILURE** ステータスになっていました。このルールは、OpenShift 4.18 に含まれる RFC 8446 の新しい暗号をチェックするように更新されました。このルールは正しく評価されるようになりました。(OCPBUGS-54212)
- 以前は、Compliance Operator プラットフォームのスキャンが失敗し、**failed to parse Ignition config** というメッセージが生成されていました。このリリースにより、バージョン 4.19 の OpenShift がお客様に提供されたときに、そのバージョンのクラスター上で Compliance Operator を安全に実行できるようになりました。(OCPBUGS-54403)
- この Compliance Operator のリリース前は、いくつかのルールがプラットフォームを認識せず、不要なエラーが発生していました。これらのルールは、他のアーキテクチャーに適切に移植されたため、正しく実行されるようになりました。ユーザーが使用しているアーキテクチャーに応じて、コンプライアンスチェック結果に **NOT-APPLICABLE** と適切に表示されます。(OCPBUGS-53041)
- 以前は、ルール **file-groupowner-ovs-conf-db-hugetlbf** が予期せず失敗していました。このリリースでは、ルールの失敗が必要な結果である場合にのみ発生するようになりました。(OCPBUGS-55190)

5.2.4. OpenShift Compliance Operator 1.6.2

OpenShift Compliance Operator 1.6.2 については、次のアドバイザリーが利用できます。

- RHBA-2025:2659 - OpenShift Compliance Operator 1.6.2 update

CVE-2024-45338 は、Compliance Operator 1.6.2 リリースで解決されています。(CVE-2024-45338)

5.2.5. OpenShift Compliance Operator 1.6.1

OpenShift Compliance Operator 1.6.1 については、次のアドバイザリーが利用できます。

- RHBA-2024:10367 - OpenShift Compliance Operator 1.6.1 update

この更新には、基礎となるベースイメージの依存関係のアップグレードが含まれています。

5.2.6. OpenShift Compliance Operator 1.6.0

OpenShift Compliance Operator 1.6.0 については、次のアドバイザリーが利用できます。

- RHBA-2024:6761 - OpenShift Compliance Operator 1.6.0 のバグ修正と機能拡張の更新

5.2.6.1. 新機能および機能拡張

- Compliance Operator には現在、Payment Card Industry Data Security Standard (PCI-DSS) バージョン 4 でサポートされているプロファイルが含まれています。詳細は、サポートされているコンプライアンスプロファイル を参照してください。
- Compliance Operator には現在、Defense Information Systems Agency Security Technical Implementation Guide (DISA STIG) V2R1 でサポートされているプロファイルが含まれています。詳細は、サポートされているコンプライアンスプロファイル を参照してください。
- **x86**、**ppc64le**、および **s390x** アーキテクチャーにインストールされた Compliance Operator で、**must-gather** 拡張が利用できるようになりました。**must-gather** ツールは、Red Hat Customer Support チームおよびエンジニアリングチームに重要な設定の詳細を提供するものです。詳細は、Compliance Operator の must-gather ツールの使用 を参照してください。

5.2.6.2. バグ修正

- このリリース前は、**ocp4-route-ip-whitelist** ルールの誤解を招くような説明により誤解が生じ、誤った設定が発生する可能性がありました。この更新により、ルールがより明確に定義されるようになりました。(CMP-2485)
- 以前は、**DONE** ステータスの **ComplianceScan** のすべての **ComplianceCheckResults** のレポートが不完全でした。この更新により、ステータスが **DONE** である **ComplianceScan** の **ComplianceCheckResults** の合計数を報告するためのアノテーションが追加されました。(CMP-2615)
- 以前は、**ocp4-cis-scc-limit-container-allowed-capabilities** ルールの説明に曖昧なガイドラインが含まれていたため、ユーザーの間で混乱が生じていました。この更新により、ルールの説明と実行可能な手順が明確になりました。(OCPBUGS-17828)
- この更新前は、sysctl 設定により、RHCOS4 ルールの特定の自動修復が、影響を受けるクラスターでスキャンに失敗していました。この更新により、正しい sysctl 設定が適用され、FedRAMP High プロファイルの RHCOS4 ルールが正しくスキャンされるようになりました。(OCPBUGS-19690)
- この更新前は、**jq** フィルターの問題により、コンプライアンスチェック中に **rhacs-operator-controller-manager** デプロイメントでエラーが発生していました。この更新により、**jq** フィルター式が更新され、**rhacs-operator-controller-manager** デプロイメントはコンテナーリソース制限に関するコンプライアンスチェックから免除され、誤検知の結果が排除されます。(OCPBUGS-19690)
- この更新前は、**rhcos4-high** および **rhcos4-moderate** プロファイルは、誤ったタイトルの設定ファイルの値をチェックしていました。その結果、一部のスキャンチェックが失敗する可能性がありました。この更新により、**rhcos4** プロファイルは正しい設定ファイルをチェックし、正しくスキャンされるようになりました。(OCPBUGS-31674)
- 以前は、**oauthclient-inactivity-timeout** ルールで使用される **accessokenInactivityTimeoutSeconds** 変数はイミュータブルだったため、DISA STIG スキャンを実行すると **FAIL** ステータスになりました。この更新により、**accessTokenInactivityTimeoutSeconds** 変数の適切な適用が正しく動作し、**PASS** ステータスが可能になりました。(OCPBUGS-32551)
- この更新前は、ルールの一部のアノテーションが更新されず、誤った制御標準が表示されていました。この更新により、ルールのアノテーションが正しく更新され、正しい制御標準が表示されるようになります。(OCPBUGS-34982)

- 以前は、Compliance Operator 1.5.1 にアップグレードすると、**ServiceMonitor** 設定で誤って参照されたシークレットが原因で、Prometheus Operator とのインテグレーションの問題が発生していました。この更新により、Compliance Operator は **ServiceMonitor** メトリクスのトークンを含むシークレットを正確に参照するようになります。(OCPBUGS-39417)

5.2.7. OpenShift Compliance Operator 1.5.1

OpenShift Compliance Operator 1.5.1 については、次のアドバイザリーが利用できます。

- RHBA-2024:5956 - OpenShift Compliance Operator 1.5.1 のバグ修正と機能拡張の更新

5.2.8. OpenShift Compliance Operator 1.5.0

OpenShift Compliance Operator 1.5.0 については、次のアドバイザリーが利用できます。

- RHBA-2024:3533 - OpenShift Compliance Operator 1.5.0 のバグ修正と機能拡張の更新

5.2.8.1. 新機能および機能拡張

- この更新により、Compliance Operator はプログラムで容易に使用できる一意のプロファイル ID を提供します。(**CMP-2450**)

- このリリースでは、Compliance Operator が ROSA HCP 環境でテストされ、サポートされるようになりました。Compliance Operator は、ROSA HCP 上で実行されるとノードプロファイルのみを読み込みます。これは、Red Hat が管理するプラットフォームではコントロールプレーンへのアクセスが制限され、Platform プロファイルが Operator の機能とは無関係になるためです (**CMP-2581**)。

5.2.8.2. バグ修正

- CVE-2024-2961 は、Compliance Operator 1.5.0 リリースで解決されています。(**CVE-2024-2961**)

- 以前は、ROSA HCP システムにおけるプロファイルリストが正しくありませんでした。この更新により、Compliance Operator のプロファイル出力が正しくなりました。(**OCPBUGS-34535**)

- このリリースでは、カスタマイズされたプロファイルで **ocp4-var-network-policies-namespaces-exempt-regex** 変数を設定することにより、namespace を **ocp4-configure-network-policies-namespaces** チェックから除外できるようになりました。(**CMP-2543**)

5.2.9. OpenShift Compliance Operator 1.4.1

OpenShift Compliance Operator 1.4.1 については、次のアドバイザリーが利用できます。

- RHBA-2024:1830 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.9.1. 新機能および機能拡張

- このリリース以降、Compliance Operator は CIS OpenShift 1.5.0 プロファイルルールを提供するようになりました。(**CMP-2447**)

- この更新により、Compliance Operator がプロファイルルールとともに **OCP4 STIG ID** と **SRG** を提供するようになりました。(**CMP-2401**)

- この更新により、**s390x** に適用されていた古いルールが削除されました。(CMP-2471)

5.2.9.2. バグ修正

- 以前は、Red Hat Enterprise Linux (RHEL) 9 を使用する Red Hat Enterprise Linux CoreOS (RHCOS) システムでは、**ocp4-kubelet-enable-protect-kernel-sysctl-file-exist** ルールの適用に失敗していました。この更新により、ルールが **ocp4-kubelet-enable-protect-kernel-sysctl** に置き換えられました。自動修復が適用されると、RHEL9 ベースの RHCOS システムでは、このルールの適用時に **PASS** と表示されるようになりました。(OCPBUGS-13589)

- 以前は、プロファイル **rhcos4-e8** を使用してコンプライアンス修正を適用した後、コアユーザーアカウントへの SSH 接続を使用してノードにアクセスできなくなりました。この更新により、ノードは `sshkey1 オプションを使用して SSH 経由でアクセス可能になります。(OCPBUGS-18331)

- 以前は、**STIG** プロファイルに、OpenShift Container Platform 用に公開された **STIG** の要件を満たす CaC のルールがありませんでした。この更新により、修復時に、Compliance Operator を使用して修復できる **STIG** 要件をクラスターが満たすようになります。(OCPBUGS-26193)

- 以前は、複数の製品に対して異なるタイプのプロファイルを持つ **ScanSettingBinding** オブジェクトを作成すると、バインディング内の複数の製品タイプに対する制限が回避されていました。この更新により、**ScanSettingBinding** オブジェクトのプロファイルタイプに関係なく、製品検証で複数の製品が許可されるようになりました。(OCPBUGS-26229)

- 以前は、自動修復が適用された後でも、**rhcos4-service-debug-shell-disabled** ルールを実行すると、**FAIL** と表示されていました。この更新により、自動修復が適用された後に **rhcos4-service-debug-shell-disabled** ルールを実行すると、**PASS** と表示されるようになりました。(OCPBUGS-28242)

- この更新により、**rhcos4-banner-etc-issue** ルールの使用手順が強化され、より詳細な情報が提供されるようになりました。(OCPBUGS-28797)

- 以前は、**api_server_api_priority_flowschema_catch_all** ルールが OpenShift Container Platform 4.16 クラスターで **FAIL** ステータスを示していました。この更新により、**api_server_api_priority_flowschema_catch_all** ルールが OpenShift Container Platform 4.16 クラスターで **PASS** ステータスを示すようになりました。(OCPBUGS-28918)

- 以前は、**ScanSettingBinding** (SSB) オブジェクトに表示される完了したスキャンからプロファイルを削除した場合、Compliance Operator が古いスキャンを削除しませんでした。その後、削除したプロファイルを使用して新しい SSB を起動すると、Compliance Operator が結果を更新できませんでした。この Compliance Operator のリリースでは、新しい SSB に新しいコンプライアンスチェック結果が表示されるようになりました。(OCPBUGS-29272)

- 以前は、**ppc64le** アーキテクチャーでは、メトリクスサービスが作成されませんでした。この更新により、Compliance Operator v1.4.1 を **ppc64le** アーキテクチャーにデプロイするときに、メトリクスサービスが正しく作成されるようになりました。(OCPBUGS-32797)

- 以前は、HyperShift ホステッドクラスターでは、**filter cannot iterate** 問題により、**ocp4-pci-dss profile** を使用したスキャンで回復不能なエラーが発生していました。このリリースでは、**ocp4-pci-dss** プロファイルのスキャンが **done** ステータスに達し、**Compliance** または **Non-Compliance** のテスト結果が返されます。(OCPBUGS-33067)

5.2.10. OpenShift Compliance Operator 1.4.0

OpenShift Compliance Operator 1.4.0 については、次のアドバイザリーが利用できます。

- RHBA-2023:7658 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.10.1. 新機能および機能拡張

- この更新により、デフォルトの **worker** および **master** ノードプール以外のカスタムノードプールを使用するクラスターで、Compliance Operator がそのノードプールの設定ファイルを確実に集約するための追加変数を指定する必要がなくなりました。
- ユーザーは、**ScanSetting.suspend** 属性を **True** に設定することで、スキャンスケジュールを一時停止できるようになりました。これにより、ユーザーは **ScanSettingBinding** を削除して再作成することなく、スキャンスケジュールを一時停止し、再アクティブ化できます。そのため、メンテナンス期間中のスキャンスケジュールを簡単に一時停止できます。(**CMP-2123**)
- Compliance Operator は、**Profile** カスタムリソースでオプションの **version** 属性をサポートするようになりました。(**CMP-2125**)
- Compliance Operator は、**ComplianceRules** のプロファイル名をサポートするようになりました。(**CMP-2126**)
- このリリースでは、**cronjob** API が改良された Compliance Operator の互換性を使用できます。(**CMP-2310**)

5.2.10.2. バグ修正

- 以前は、Windows ノードがコンプライアンススキャンでスキップされなかったため、Windows ノードを含むクラスターでは、自動修復の適用後に一部のルールが失敗していました。このリリースでは、スキャン時に Windows ノードが正しくスキップされます。(**OCPBUGS-7355**)
- この更新により、**rprivate** のデフォルトのマウント伝播が、マルチパス構成に依存する Pod のルートボリュームマウントに対して正しく処理されるようになりました。(**OCPBUGS-17494**)
- 以前は、Compliance Operator は、修復の適用中であってもルールを調整せずに **coreos_vsyscall_kernel_argument** の修復を生成していました。リリース 1.4.0 では、**coreos_vsyscall_kernel_argument** ルールがカーネル引数を適切に評価し、適切な修復を生成するようになりました。(**OCPBUGS-8041**)
- この更新より前は、自動修復が適用された後でも、**rhcos4-audit-rules-login-events-faillock** のルールが失敗していました。この更新により、**rhcos4-audit-rules-login-events-faillock** の失敗ロックが自動修復後に正しく適用されるようになりました。(**OCPBUGS-24594**)
- 以前は、Compliance Operator 1.3.1 から Compliance Operator 1.4.0 にアップグレードすると、OVS ルールのスキャン結果が **PASS** から **NOT-APPLICABLE** に変わりました。この更新により、OVS ルールのスキャン結果に **PASS** が表示されるようになりました (**OCPBUGS-25323**)

5.2.11. OpenShift Compliance Operator 1.3.1

OpenShift Compliance Operator 1.3.1 については、次のアドバイザリーが利用できます。

- RHBA-2023:5669 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

この更新は、基礎となる依存関係の CVE に対処するものです。

5.2.11.1. 新機能および機能拡張

- Compliance Operator は、FIPS モードで実行されている OpenShift Container Platform クラスターにインストールして使用できます。

重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL で FIPS モードを設定する方法の詳細は、RHEL から FIPS モードへの切り替え を参照してください。

FIPS モードでブートされた Red Hat Enterprise Linux (RHEL) または Red Hat Enterprise Linux CoreOS (RHCOS) を実行する場合、OpenShift Container Platform コアコンポーネントは、x86_64、ppc64le、および s390x アーキテクチャーのみで、FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。

5.2.11.2. 既知の問題

- Windows ノードがコンプライアンススキャンでスキップされないため、Windows ノードを含むクラスターでは、自動修復の適用後に一部のルールが失敗します。スキャン時に Windows ノードをスキップする必要があるため、これは想定される結果とは異なります。(**OCPBUGS-7355**)

5.2.12. OpenShift Compliance Operator 1.3.0

OpenShift Compliance Operator 1.3.0 については、次のアドバイザリーが利用できます。

- RHBA-2023:5102 - OpenShift Compliance Operator の機能拡張の更新

5.2.12.1. 新機能および機能拡張

- OpenShift Container Platform の国防情報システム局セキュリティ技術導入ガイド (DISA-STIG) が、Compliance Operator 1.3.0 から利用できるようになりました。詳細は、サポート対象のコンプライアンスプロファイル を参照してください。

- Compliance Operator 1.3.0 は、NIST 800-53 Moderate-Impact Baseline for OpenShift Container Platform のプラットフォームおよびノードプロファイルに対応する IBM Power® および IBM Z® をサポートするようになりました。

5.2.13. OpenShift Compliance Operator 1.2.0

OpenShift Compliance Operator 1.2.0 については、次のアドバイザリーが利用できます。

- RHBA-2023:4245 - OpenShift Compliance Operator の機能拡張の更新

5.2.13.1. 新機能および機能拡張

- CIS OpenShift Container Platform 4 Benchmark v1.4.0 プロファイルがプラットフォームおよびノードアプリケーションで利用できるようになりました。CIS OpenShift Container Platform v4 ベンチマークを見つけるには、CIS Benchmarks にアクセスし、**Download Latest CIS Benchmark** をクリックします。そこで登録すると、ベンチマークをダウンロードできます。

重要

Compliance Operator 1.2.0 にアップグレードすると、CIS OpenShift Container Platform 4 Benchmark 1.1.0 プロファイルが上書きされます。

OpenShift Container Platform 環境に既存の **cis** および **cis-node** 修復が含まれている場合は、Compliance Operator 1.2.0 にアップグレードした後のスキャン結果に多少の違いが生じる可能性があります。

- **scc-limit-container-allowed-capabilities** ルールで、Security Context Constraints (SCC) の監査に関する明確性がさらに向上しました。

5.2.14. OpenShift Compliance Operator 1.1.0

OpenShift Compliance Operator 1.1.0 には、以下のアドバイザリーを利用できます。

- RHBA-2023:3630 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.14.1. 新機能および機能拡張

- **ComplianceScan** カスタムリソース定義 (CRD) ステータスで開始および終了のタイムスタンプが利用できるようになりました。
- **Subscription** ファイルを作成することで、OperatorHub を使用して Compliance Operator を Hosted Control Plane にデプロイできるようになりました。詳細は、Hosted Control Plane への Compliance Operator のインストール を参照してください。

5.2.14.2. バグ修正

- 今回の更新前は、Compliance Operator ルールの指示の一部が存在していませんでした。今回の更新後、以下のルールの手順が改善されました。
  - **classification_banner**
  - **oauth_login_template_set**
  - **oauth_logout_url_set**
  - **oauth_provider_selection_set**
  - **ocp_allowed_registries**
  - **ocp_allowed_registries_for_import** (OCPBUGS-10473)
- この更新前は、チェックの精度とルールの指示が不明確でした。今回の更新後、次の **sysctl** ルールのチェック精度と手順が改善されました。
  - **kubelet-enable-protect-kernel-sysctl**
  - **kubelet-enable-protect-kernel-sysctl-kernel-keys-root-maxbytes**
  - **kubelet-enable-protect-kernel-sysctl-kernel-keys-root-maxkeys**
  - **kubelet-enable-protect-kernel-sysctl-kernel-panic**

  - **kubelet-enable-protect-kernel-sysctl-kernel-panic-on-oops**
  - **kubelet-enable-protect-kernel-sysctl-vm-overcommit-memory**
  - **kubelet-enable-protect-kernel-sysctl-vm-panic-on-oom** (OCPBUGS-11334)

- 今回の更新以前は、**ocp4-alert-receiver-configured** ルールに命令が含まれていませんでした。今回の更新により、**ocp4-alert-receiver-configured** ルールに含まれる命令が改善されました。(OCPBUGS-7307)

- 今回の更新より前は、**rhcos4-sshd-set-loglevel-info** ルールは **rhcos4-e8** プロファイルでは失敗していました。この更新により、**sshd-set-loglevel-info** ルールの修復が更新され、正しい設定変更が適用され、修復の適用後に後続のスキャンが通過できるようになりました。(OCPBUGS-7816)

- 今回の更新の前は、最新の Compliance Operator インストールを使用した OpenShift Container Platform の新規インストールは、**scheduler-no-bind-address** ルールで失敗していました。今回の更新では、パラメーターが削除されたので、OpenShift Container Platform の新しいバージョンでは **scheduler-no-bind-address** ルールが無効になりました。(OCPBUGS-8347)

5.2.15. OpenShift Compliance Operator 1.0.0

OpenShift Compliance Operator 1.0.0 には、以下のアドバイザリーを利用できます。

- RHBA-2023:1682 - OpenShift Compliance Operator バグ修正更新

5.2.15.1. 新機能および機能拡張

- Compliance Operator は stable になり、リリースチャネルは **stable** にアップグレードされました。将来のリリースは Semantic Versioning に従います。最新リリースにアクセスするには、コンプライアンス Operator の更新 を参照してください。

5.2.15.2. バグ修正

- この更新前は、compliance_operator_compliance_scan_error_total メトリックには、エラーメッセージごとに異なる値を持つ ERROR ラベルがありました。この更新により、compliance_operator_compliance_scan_error_total メトリックの値は増加しません。(OCPBUGS-1803)

- この更新前は、**ocp4-api-server-audit-log-maxsize** ルールにより **FAIL** 状態が発生していました。この更新により、エラーメッセージがメトリックから削除され、ベストプラクティスに従ってメトリックのカーディナリティが減少しました。(OCPBUGS-7520)

- この更新前は、**rhcos4-enable-fips-mode** ルールの説明が分かりにくく、インストール後に FIPS を有効化できるという誤解を招くものでした。この更新により、**rhcos4-enable-fips-mode** ルールの説明で、インストール時に FIPS を有効化する必要があることを明確に示されました。(OCPBUGS-8358)

5.2.16. OpenShift Compliance Operator 0.1.61

OpenShift Compliance Operator 0.1.61 には、以下のアドバイザリーを利用できます。

- RHBA-2023:0557 - OpenShift Compliance Operator bug fix update

5.2.16.1. 新機能および機能拡張

- Compliance Operator は、スキャナー Pod のタイムアウト設定をサポートするようになりました。タイムアウトは **ScanSetting** オブジェクトで指定されます。スキャンがタイムアウト内に完了しない場合、スキャンは最大再試行回数に達するまで再試行します。詳細は、ScanSetting タイムアウトの設定 を参照してください。

5.2.16.2. バグ修正

- 今回の更新以前は、Compliance Operator は必要な変数を入力として修復していました。変数が設定されていない修復はクラスター全体に適用され、修復が正しく適用された場合でも、ノードがスタックしていました。今回の更新により、Compliance Operator は、修復のために **TailoredProfile** を使用して変数を提供する必要があるかどうかを検証します。(OCPBUGS-3864)

- 今回の更新以前は、**ocp4-kubelet-configure-tls-cipher-suites** の手順が不完全であるため、ユーザーはクエリーを手動で調整する必要がありました。今回の更新により、**ocp4-kubelet-configure-tls-cipher-suites** で提供されるクエリーが実際の結果を返し、監査ステップを実行します。(OCPBUGS-3017)

- 今回の更新以前は、システム予約パラメーターが kubelet 設定ファイルで生成されず、Compliance Operator はマシン設定プールの一時停止に失敗していました。今回の更新により、Compliance Operator は、マシン設定プールの評価時にシステム予約パラメーターを省略します。(OCPBUGS-4445)

- 今回の更新以前は、**ComplianceCheckResult** オブジェクトに正しい説明がありませんでした。今回の更新により、Compliance Operator は、ルールの説明から **ComplianceCheckResult** 情報を取得します。(OCPBUGS-4615)

- この更新前は、Compliance Operator はマシン設定の解析時に空の kubelet 設定ファイルをチェックしませんでした。その結果、Compliance Operator はパニックになり、クラッシュします。今回の更新により、Compliance Operator は kubelet 設定データ構造の改善されたチェックを実装し、完全にレンダリングされた場合にのみ続行します。(OCPBUGS-4621)

- この更新前は、Compliance Operator がマシン設定プール名と猶予期間に基づいて kubelet の退避に対する修復を生成していました。そのため、1 つの退避ルールに対して複数の修復が発生していました。この更新により、Compliance Operator は 1 つのルールに対してすべての修復を適用するようになりました。(OCPBUGS-4338)

- この更新の前に、デフォルト以外の **MachineConfigPool** で **TailoredProfile** を使用する **ScanSettingBinding** を作成しようとすると、**ScanSettingBinding** が **Failed** としてマークされるというリグレッションが発生していました。今回の更新で、機能が復元され、**TailoredProfile** を使用してカスタム **ScanSettingBinding** が正しく実行されるようになりました。(OCPBUGS-6827)

- 今回の更新以前は、一部の kubelet 設定パラメーターにデフォルト値がありませんでした。今回の更新により、次のパラメーターにデフォルト値が含まれます (OCPBUGS-6708)。
    - **ocp4-cis-kubelet-enable-streaming-connections**
    - **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-available**
    - **ocp4-cis-kubelet-eviction-thresholds-set-hard-imagefs-inodesfree**

  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-memory-available**
  - **ocp4-cis-kubelet-eviction-thresholds-set-hard-nodefs-available**

- 今回の更新以前は、kubelet の実行に必要なパーミッションが原因で、**selinux_confinement_of_daemons** ルールが kubelet で実行できませんでした。今回の更新で、**selinux_confinement_of_daemons** ルールが無効になります。(OCPBUGS-6968)

5.2.17. OpenShift Compliance Operator 0.1.59

OpenShift Compliance Operator 0.1.59 については、次のアドバイザリーが利用できます。

- RHBA-2022:8538 - OpenShift Compliance Operator バグ修正更新

5.2.17.1. 新機能および機能拡張

- Compliance Operator は、**ppc64le** アーキテクチャーで Payment Card Industry Data Security Standard (PCI-DSS) **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートするようになりました。

5.2.17.2. バグ修正

- 以前は、Compliance Operator は **ppc64le** などの異なるアーキテクチャーで Payment Card Industry Data Security Standard (PCI DSS) **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートしていませんでした。Compliance Operator は、**ppc64le** アーキテクチャーで **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルをサポートするようになりました。(OCPBUGS-3252)

- 以前は、バージョン 0.1.57 への最近の更新後、**rerunner** サービスアカウント (SA) はクラスターサービスバージョン (CSV) によって所有されなくなり、Operator のアップグレード中に SA が削除されました。現在、CSV は 0.1.59 で **rerunner** SA を所有しており、以前のバージョンからアップグレードしても SA が欠落することはありません。(OCPBUGS-3452)

5.2.18. OpenShift Compliance Operator 0.1.57

OpenShift Compliance Operator 0.1.57 には、以下のアドバイザリーを利用できます。

- RHBA-2022:6657 - OpenShift Compliance Operator バグ修正更新

5.2.18.1. 新機能および機能拡張

- **KubeletConfig** チェックが **Node** から **Platform** タイプに変更されました。**KubeletConfig** は、**KubeletConfig** のデフォルト設定を確認します。設定ファイルは、すべてのノードからノードプールごとに 1 つの場所に集約されます。デフォルトの設定値をもとにした **KubeletConfig** ルールの評価 を参照してください。

- **ScanSetting** カスタムリソースでは、**scanLimits** 属性を使用して、スキャナー Pod のデフォルトの CPU およびメモリー制限をオーバーライドできるようになりました。詳細は、Compliance Operator のリソース制限の引き上げ を参照してください。

- **PriorityClass** オブジェクトは **ScanSetting** で設定できるようになりました。これにより、Compliance Operator が優先され、クラスターがコンプライアンス違反になる可能性が最小限に抑えられます。詳細は、**ScanSetting** スキャンの **PriorityClass** の設定 を参照してください。

5.2.18.2. バグ修正

- 以前のバージョンでは、Compliance Operator は通知をデフォルトの **openshift-compliance** namespace にハードコーディングしていました。Operator がデフォルト以外の namespace にインストールされている場合、通知は予想通りに機能しませんでした。今回のリリースより、通知はデフォルト以外の **openshift-compliance** namespace で機能するようになりました。(BZ#2060726)
- 以前のバージョンでは、Compliance Operator は kubelet オブジェクトによって使用されるデフォルト設定を評価できず、不正確な結果と誤検出が発生していました。この新機能 は kubelet 設定を評価し、正確に報告するようになりました。(BZ#2075041)
- 以前は、Compliance Operator は、**eventRecordQPS** の値がデフォルト値よりも大きいため、自動修復の適用後に **FAIL** 状態で **ocp4-kubelet-configure-event-creation** ルールを報告していました。**ocp4-kubelet-configure-event-creation** ルール修復はデフォルト値を設定し、ルールが正しく適用されるようになりました。(BZ#2082416)
- **ocp4-configure-network-policies** ルールでは、効果的に実行するために手動の介入が必要です。新しい説明的な指示とルールの更新により、Calico CNI を使用するクラスターに対する **ocp4-configure-network-policies** ルールの適用性が向上します。(BZ#2091794)
- 以前は、スキャン設定で **debug=true** オプションを使用すると、Compliance Operator がインフラストラクチャーのスキャンに使用される Pod をクリーンアップしませんでした。これにより、**ScanSettingBinding** を削除した後でも Pod がクラスターに残されました。現在、**ScanSettingBinding** が削除されると、Pod は常に削除されます (BZ#2092913)。
- 以前のバージョンでは、Compliance Operator は古いバージョンの **operator-sdk** コマンドを使用しており、非推奨の機能についてアラートが発生していました。現在、**operator-sdk** コマンドの更新バージョンが含まれており、廃止された機能に関するアラートが発生することはなくなりました。(BZ#2098581)
- 以前のバージョンでは、Compliance Operator は kubelet とマシン設定の関係を判別できない場合に、修復の適用に失敗しました。Compliance Operator はマシン設定の処理を改善し、kubelet 設定がマシン設定のサブセットであるかどうかを判別できるようになりました。(BZ#2102511)
- 以前のリリースでは、**ocp4-cis-node-master-kubelet-enable-cert-rotation** のルールで成功基準が適切に記述されていませんでした。その結果、**RotateKubeletClientCertificate** の要件が不明確でした。今回のリリースでは、**ocp4-cis-node-master-kubelet-enable-cert-rotation** のルールで、kubelet 設定ファイルにある設定に関係なく正確にレポートされるようになりました。(BZ#2105153)
- 以前のリリースでは、アイドルストリーミングタイムアウトをチェックするルールでデフォルト値が考慮されなかったため、ルールレポートが正確ではありませんでした。今回のリリースより、チェックがより厳密に行われるようになり、デフォルトの設定値に基づいて結果の精度が向上しました。(BZ#2105878)
- 以前のバージョンでは、Compliance Operator は Ignition 仕様なしでマシン設定を解析する際に API リソースを取得できず、**api-check-pods** プロセスがクラッシュしていました。現在は、Compliance Operator は Ignition 仕様を持たないマシン設定プールを正しく処理します。(BZ#2117268)
- 以前のリリースでは、**modprobe** 設定の値が一致しないことが原因で、修復の適用後に **modprobe** 設定を評価するルールが失敗することがありました。今回のリリースより、チェックおよび修復の **modprobe** 設定に同じ値が使用されるようになり、結果の一貫性が保たれるようになりました。(BZ#2117747)

5.2.18.3. 非推奨

- Install into all namespacesin the clusterを指定するか、**WATCH_NAMESPACES** 環境変数を **""** に設定しても、すべての namespace に設定が適用されなくなりました。Compliance Operator のインストール時に指定されていない namespace にインストールされた API リソースは動作しなくなります。API リソースでは、選択した namespace またはデフォルトで **openshift-compliance** namespace を作成する必要がある場合があります。今回の変更により、Compliance Operator のメモリー使用量が改善されます。

5.2.19. OpenShift Compliance Operator 0.1.53

OpenShift Compliance Operator 0.1.53 については、次のアドバイザリーが利用できます。

- RHBA-2022:5537 - OpenShift Compliance Operator bug fix update

5.2.19.1. バグ修正

- 以前は、**ocp4-kubelet-enable-streaming-connections** ルールに誤った変数比較が含まれていたため、スキャン結果が誤検出されていました。現在、Compliance Operator は、**streamingConnectionIdleTimeout** を設定するときに正確なスキャン結果を提供します。(BZ#2069891)

- 以前は、**/etc/openvswitch/conf.db** のグループ所有権が IBM Z® アーキテクチャーで正しくなかったため、**ocp4-cis-node-worker-file-groupowner-ovs-conf-db** のチェックが失敗していました。現在、このチェックは IBM Z® アーキテクチャーシステムでは **NOT-APPLICABLE** とマークされています。(BZ#2072597)

- 以前は、デプロイメント内の Security Context Constraints (SCC) ルールに関するデータが不完全なため、**ocp4-cis-scc-limit-container-allowed-capabilities** ルールが **FAIL** 状態で報告されていました。現在は、結果は **MANUAL** ですが、これは、人間の介入を必要とする他のチェックと一致しています。(BZ#2077916)

- 以前は、以下のルールが API サーバーおよび TLS 証明書とキーの追加の設定パスを考慮していなかったため、証明書とキーが適切に設定されていても失敗が報告されていました。

  - **ocp4-cis-api-server-kubelet-client-cert**
  - **ocp4-cis-api-server-kubelet-client-key**
  - **ocp4-cis-kubelet-configure-tls-cert**
  - **ocp4-cis-kubelet-configure-tls-key**

  これで、ルールは正確にレポートし、kubelet 設定ファイルで指定されたレガシーファイルパスを監視します。(BZ#2079813)

- 以前は、**content_rule_oauth_or_oauthclient_inactivity_timeout** ルールは、タイムアウトのコンプライアンスを評価するときに、デプロイメントによって設定された設定可能なタイムアウトを考慮していませんでした。その結果、タイムアウトが有効であってもルールが失敗していました。現在、Compliance Operator は、**var_oauth_inactivity_timeout** 変数を使用して、有効なタイムアウトの長さを設定しています。(BZ#2081952)

- 以前は、Compliance Operator は、特権使用に適切にラベル付けされていない namespace に対して管理者権限を使用していたため、Pod のセキュリティーレベル違反に関する警告メッセージが表示されていました。現在、Compliance Operator は、適切な namespace ラベルと権限調

整を行い、権限に違反することなく結果にアクセスできるようになっています。(BZ#2088202)

- 以前は、**rhcos4-high-master-sysctl-kernel-yama-ptrace-scope** および **rhcos4-sysctl-kernel-core-pattern** に自動修復を適用すると、それらのルールが修復されても、その後のスキャン結果で失敗することがありました。現在、修復が適用された後でも、ルールは **PASS** を正確に報告します。(BZ#2094382)
- 以前は、Compliance Operator は、メモリー不足の例外が原因で **CrashLoopBackoff** 状態で失敗していました。現在では、Compliance Operator は、メモリー内の大規模なマシン設定データセットを処理し、正しく機能するように改良されました。(BZ#2094854)

5.2.19.2. 既知の問題

- **ScanSettingBinding** オブジェクト内で **"debug":true** が設定されている場合に、そのバインディングが削除されても、**ScanSettingBinding** オブジェクトによって生成された Pod は削除されません。回避策として、次のコマンドを実行して残りの Pod を削除します。

  ```
  $ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
  ```

  (BZ#2092913)

5.2.20. OpenShift Compliance Operator 0.1.52

OpenShift Compliance Operator 0.1.52 については、次のアドバイザリーが利用できます。

- RHBA-2022:4657 - OpenShift Compliance Operator bug fix update

5.2.20.1. 新機能および機能拡張

- FedRAMP の SCAP プロファイル high が、OpenShift Container Platform 環境で使用できるようになりました。詳細は、サポートされているコンプライアンスプロファイル を参照してください。

5.2.20.2. バグ修正

- 以前は、**DAC_OVERRIDE** 機能が削除されているセキュリティー環境でのマウントパーミッションの問題が原因で、**OpenScap** コンテナーがクラッシュしていました。今回は、実行可能なマウントパーミッションがすべてのユーザーに適用されるようになりました。(BZ#2082151)
- 以前は、コンプライアンスルール **ocp4-configure-network-policies** を **MANUAL** として設定できました。今回は、コンプライアンスルール **ocp4-configure-network-policies** が **AUTOMATIC** に設定されるようになりました。(BZ#2072431)
- 以前は、Compliance Operator のスキャン Pod はスキャン後に削除されないため、Cluster Autoscaler はスケールダウンできませんでした。今回は、デバッグ目的で明示的に保存されていない限り、Pod はデフォルトで各ノードから削除されるようになりました。(BZ#2075029)
- 以前は、Compliance Operator を **KubeletConfig** に適用すると、マシン設定プールの一時停止が早すぎてノードが **NotReady** 状態になることがありました。今回は、マシン設定プールは停止されず、ノードが正常に機能するようになりました。(BZ#2071854)
- 以前のバージョンでは、Machine Config Operator は最新のリリースで **url-encoded** コードではなく **base64** を使用していたため、Compliance Operator の修復が失敗していました。

Compliance Operator は、**base64** および **url-encoded** マシン設定コードの両方を処理するようにエンコーディングをチェックし、修復が適切に実行されるようになりました。(BZ#2082431)

5.2.20.3. 既知の問題

- **ScanSettingBinding** オブジェクト内で **"debug":true** が設定されている場合に、そのバインディングが削除されても、**ScanSettingBinding** オブジェクトによって生成された Pod は削除されません。回避策として、次のコマンドを実行して残りの Pod を削除します。

```
$ oc delete pods -l compliance.openshift.io/scan-name=ocp4-cis
```

  (BZ#2092913)

5.2.21. OpenShift Compliance Operator 0.1.49

OpenShift Compliance Operator 0.1.49 については、次のアドバイザリーが利用できます。

- RHBA-2022:1148 - OpenShift Compliance Operator bug fix and enhancement update

5.2.21.1. 新機能および機能拡張

- Compliance Operator は、次のアーキテクチャーでサポートされるようになりました。
  - IBM Power®
  - IBM Z®
  - IBM® LinuxONE

5.2.21.2. バグ修正

- 以前は、**openshift-compliance** コンテンツには、ネットワークタイプのプラットフォーム固有のチェックが含まれていませんでした。その結果、OVN および SDN 固有のチェックは、ネットワーク設定に基づいて **not-applicable** ではなく、**failed** したものとして表示されます。現在、新しいルールにはネットワークルールのプラットフォームチェックが含まれているため、ネットワーク固有のチェックをより正確に評価できます。(BZ#1994609)

- 以前は、**ocp4-moderate-routes-protected-by-tls** ルールは TLS 設定を誤ってチェックしていたため、接続がセキュアな SSL/TLS プロトコルであっても、ルールがチェックに失敗していました。現在、このチェックでは、ネットワークガイダンスおよびプロファイルの推奨事項と一致する TLS 設定が適切に評価されます。(BZ#2002695)

- 以前は、**ocp-cis-configure-network-policies-namespace** は、namespaces を要求するときにページネーションを使用していました。これにより、デプロイメントで 500 を超える namespaces のリストが切り捨てられたため、ルールが失敗しました。今回のバージョンでは、namespace の全リストが必要になり、namespace が 500 以上ある場合でも、設定済みのネットワークポリシーをチェックするルールが機能するようになりました。(BZ#2038909)

- 以前は、**sshd jinja** マクロを使用した修復は、特定の sshd 設定にハードコーディングされていました。その結果、設定がルールがチェックしているコンテンツと一致せず、チェックが失敗していました。これで、sshd 設定がパラメーター化され、ルールが正常に適用されます。(BZ#2049141)

- 以前は、**ocp4-cluster-version-operator-verify-integrity** が、常に Cluter Version Operator

(CVO) 履歴の最初のエントリーをチェックしていました。その結果、後続の OpenShift Container Platform バージョンが検証される状況ではアップグレードが失敗します。これで、**ocp4-cluster-version-operator-verify-integrity** のコンプライアンスチェック結果は、検証済みバージョンを検出でき、CVO 履歴で正確になります。(BZ#2053602)

- 以前は、**ocp4-api-server-no-adm-ctrl-plugins-disabled** ルールは、空のアドミッションコントローラープラグインのリストをチェックしませんでした。その結果、すべてのアドミッションプラグインが有効になっている場合でも、ルールは常に失敗します。今回のリリースでは、**ocp4-api-server-no-adm-ctrl-plugins-disabled** ルールのチェックが強力になり、すべての受付コントローラープラグインを有効にすると、問題なく成功するようになりました。(BZ#2058631)

- 以前は、スキャンには Linux ワーカーノードに対して実行するためのプラットフォームチェックが含まれていませんでした。その結果、Linux ベースではないワーカーノードに対してスキャンを実行すると、スキャンループが終了することはありませんでした。今回のリリースでは、スキャンは、プラットフォームのタイプとラベルに基づいて適切にスケジュールされ、正確に実行されます。(BZ#2056911)

5.2.22. OpenShift Compliance Operator 0.1.48

OpenShift Compliance Operator 0.1.48 には、以下のアドバイザリーを利用できます。

- RHBA-2022:0416 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.22.1. バグ修正

- 以前は、拡張された Open Vulnerability and Assessment Language (OVAL) 定義に関連する一部のルールの **checkType** は **None** でした。これは、Compliance Operator が、ルールを解析するときに拡張 OVAL 定義を処理していなかったためです。この更新により、拡張 OVAL 定義のコンテンツが解析され、これらのルールの **checkType** が **Node** または **Platform** になります。(BZ#2040282)

- 以前は、**KubeletConfig** 用に手動で作成された **MachineConfig** オブジェクトにより、**KubeletConfig** オブジェクトが修復用に生成されなくなり、修復が **Pending** 状態のままになりました。このリリースでは、**KubeletConfig** 用に手動で作成された **MachineConfig** オブジェクトがあるかどうかに関係なく、修復によって **KubeletConfig** オブジェクトが作成されます。その結果、**KubeletConfig** の修正が期待どおりに機能するようになりました。(BZ#2040401)

5.2.23. OpenShift Compliance Operator 0.1.47

OpenShift Compliance Operator 0.1.47 には、以下のアドバイザリーを利用できます。

- RHBA-2022:0014 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.23.1. 新機能および機能拡張

- Compliance Operator は、Payment Card Industry Data Security Standard (PCI DSS) の次のコンプライアンスベンチマークをサポートするようになりました。
  - ocp4-pci-dss
  - ocp4-pci-dss-node

- FedRAMP の中程度の影響レベルの追加のルールと修正が、OCP4-moderate、OCP4-moderate-node、および rhcos4-moderate プロファイルに追加されます。

- KubeletConfig の修正がノードレベルのプロファイルで利用できるようになりました。

5.2.23.2. バグ修正

- 以前は、クラスターが OpenShift Container Platform 4.6 以前を実行していた場合、中程度のプロファイルでは USBGuard 関連のルールの修正が失敗していました。これは、Compliance Operator によって作成された修正が、ドロップインディレクトリーをサポートしていない古いバージョンの USBGuard に基づいていたためです。現在、OpenShift Container Platform 4.6 を実行しているクラスターでは、USBGuard 関連のルールの無効な修復は作成されません。クラスターが OpenShift Container Platform 4.6 を使用している場合は、USBGuard 関連のルールの修正を手動で作成する必要があります。
  さらに、修正は、最小バージョン要件を満たすルールに対してのみ作成されます。(BZ#1965511)

- 以前のリリースでは、修正をレンダリングする場合には、Compliance Operator は、その修正が適切に設定されているかを、厳密すぎる正規表現を使用して確認していました。その結果、**sshd_config** をレンダリングするような一部の修正は、正規表現チェックに合格しないため、作成されませんでした。正規表現は不要であることが判明し、削除されました。修復が正しくレンダリングされるようになりました。(BZ#2033009)

5.2.24. OpenShift Compliance Operator 0.1.44

OpenShift Compliance Operator 0.1.44 には、以下のアドバイザリーを利用できます。

- RHBA-2021:4530-OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.24.1. 新機能および機能拡張

- このリリースでは、**strictNodeScan** オプションが **ComplianceScan**、**ComplianceSuite**、および **ScanSetting** CR に追加されました。このオプションのデフォルトは **true** で、これは以前の動作と一致します。ノードでスキャンをスケジュールできなかった場合にエラーが発生しました。このオプションを **false** に設定すると、Compliance Operator は、スキャンのスケジュールについてより寛容になります。エフェメラルノードのある環境では、**strictNodeScan** 値を false に設定できます。これにより、クラスター内の一部のノードがスケジューリングに使用できない場合でも、コンプライアンススキャンを続行できます。

- **ScanSetting** オブジェクトの **nodeSelector** 属性と **tolerations** 属性を設定することにより、結果サーバーのワークロードをスケジュールするために使用されるノードをカスタマイズできるようになりました。これらの属性は、PV ストレージボリュームをマウントし、生の Asset Reporting Format (ARF) 結果を格納するために使用される Pod である **ResultServer** Pod を配置するために使用されます。以前は、**nodeSelector** パラメーターおよび **tolerations** パラメーターは、デフォルトでコントロールプレーンノードの1つを選択し、**node-role.kubernetes.io/master taint** を許容していました。これは、コントロールプレーンノードが PV をマウントすることを許可されていない環境では機能しませんでした。この機能は、ノードを選択し、それらの環境で異なるテイントを許容する方法を提供します。

- Compliance Operator は、**KubeletConfig** オブジェクトを修正できるようになりました。

- エラーメッセージを含むコメントが追加され、コンテンツ開発者がクラスター内に存在しないオブジェクトとフェッチできないオブジェクトを区別できるようになりました。

- ルールオブジェクトには、**checkType** および **description** の 2 つの新しい属性が含まれるようになりました。これらの属性を使用すると、ルールがノードチェックとプラットフォームチェックのどちらに関連しているかを判断したり、ルールの機能を確認したりできます。

- 今回の機能拡張により、既存のプロファイルを拡張して調整されたプロファイルを作成する必要があるという要件がなくなります。これは、**TailoredProfile** CRD の **extends** フィールドが必須ではなくなったことを意味します。これで、ルールオブジェクトのリストを選択して、調整されたプロファイルを作成できます。**compliance.openshift.io/product-type:** アノテーションを設定するか、**TailoredProfile** CR の **-node** 接尾辞を設定して、プロファイルをノードに適用するかプラットフォームに適用するかを選択する必要があることに注意してください。

- このリリースでは、Compliance Operator は、テイントに関係なく、すべてのノードでスキャンをスケジュールできるようになりました。以前は、スキャン Pod は **node-role.kubernetes.io/master taint** のみを許容していました。つまり、テイントのないノードで実行するか、**node-role.kubernetes.io/master** テイントのあるノードでのみ実行していました。ノードにカスタムテイントを使用するデプロイメントでは、これにより、それらのノードでスキャンがスケジュールされませんでした。現在、スキャン Pod はすべてのノードのテイントを許容します。

- このリリースでは、Compliance Operator は、次の North American Electric Reliability Corporation (NERC) のセキュリティープロファイルをサポートしています。
  - ocp4-nerc-cip
  - ocp4-nerc-cip-node
  - rhcos4-nerc-cip

- このリリースでは、Compliance Operator は、NIST 800-53 Moderate-Impact Baseline for the Red Hat OpenShift - Node level, ocp4-moderate-node セキュリティープロファイルをサポートします。

5.2.24.2. テンプレートと変数の使用

- このリリースでは、修復テンプレートで複数値の変数を使用できるようになりました。

- この更新により、Compliance Operator は、コンプライアンスプロファイルで設定された変数に基づいて修正を変更できます。これは、タイムアウト、NTP サーバーのホスト名などのデプロイメント固有の値を含む修復に役立ちます。さらに、**ComplianceCheckResult** オブジェクトは、チェックが使用した変数をリストするラベル **compliance.openshift.io/check-has-value** を使用するようになりました。

5.2.24.3. バグ修正

- 以前は、スキャンの実行中に、Pod のスキャナーコンテナーの 1 つで予期しない終了が発生しました。このリリースでは、Compliance Operator は、クラッシュを回避するために最新の OpenSCAP バージョン 1.3.5 を使用します。

- 以前は、**autoReplyRemediations** を使用して修復を適用すると、クラスターノードの更新がトリガーされていました。一部の修復に必要な入力変数がすべて含まれていない場合、これは中断されました。これで、修復に 1 つ以上の必要な入力変数が欠落している場合、**NeedsReview** の状態が割り当てられます。1 つ以上の修復が **NeedsReview** 状態にある場合、マシン設定プールは一時停止されたままになり、必要なすべての変数が設定されるまで修復は適用されません。これにより、ノードの中断を最小限に抑えることができます。

- Prometheus メトリックに使用される RBAC Role と Role Binding が 'ClusterRole' と 'ClusterRoleBinding' に変更なり、カスタマイズなしで監視が機能するようになりました。
- 以前は、プロファイルの解析中にエラーが発生した場合、ルールまたは変数オブジェクトが削除され、プロファイルから削除されていました。これで、解析中にエラーが発生した場合、**profileparser** はオブジェクトに一時的なアノテーションを付けて、解析が完了するまでオブジェクトが削除されないようにします。(BZ#1988259)
- 以前のバージョンでは、調整されたプロファイルにタイトルまたは説明がない場合、エラーが発生しました。XCCDF 標準では、調整されたプロファイルのタイトルと説明が必要なため、タイトルと説明を **TailoredProfile** CR で設定する必要があります。
- 以前は、調整されたプロファイルを使用する場合、特定の選択セットのみを使用して **TailoredProfile** 変数値を設定できました。この制限がなくなり、**TailoredProfile** 変数を任意の値に設定できるようになりました。

5.2.25. Compliance Operator 0.1.39 リリースノート

OpenShift Compliance Operator 0.1.39 については、次のアドバイザリーが利用できます。

- RHBA-2021:3214 - OpenShift Compliance Operator のバグ修正と機能拡張の更新

5.2.25.1. 新機能および機能拡張

- 以前は、Compliance Operator は、Payment Card Industry Data Security Standard (PCI DSS) 参照を解析できませんでした。現在、Operator は、PCI DSS プロファイルで提供されるコンプライアンスコンテンツを解析できるようになりました。
- 以前は、Compliance Operator は、中程度のプロファイルで AU-5 制御のルールを実行できませんでした。これで、**Prometheusrules.monitoring.coreos.com** オブジェクトを読み取り、中程度のプロファイルで AU-5 コントロールを扱うルールを実行できるように、Operator にアクセス許可が追加されました。

5.2.26. 関連情報

- Compliance Operator について

5.3. COMPLIANCE OPERATOR のサポート

5.3.1. Compliance Operator のライフサイクル

Compliance Operator は "Rolling Stream" Operator で、OpenShift Container Platform リリースの更新が非同期で利用可能であることを意味します。詳細は、Red Hat カスタマーポータルの OpenShift Operator ライフサイクル を参照してください。

5.3.2. サポート

このドキュメントで説明されている手順、または OpenShift Container Platform 全般で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。

カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。

- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントについて Jira 課題 を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

5.3.3. Compliance Operator の must-gather ツールの使用

Compliance Operator v1.6.0 以降では、Compliance Operator イメージで **must-gather** コマンドを実行することで、Compliance Operator リソースに関するデータを収集できます。

注記

サポートケースの作成時またはバグレポートの提出時は、Operator の設定とログに関する追加の詳細が提供される **must-gather** ツールの使用を検討してください。

手順

- Compliance Operator に関するデータを収集するには、次のコマンドを実行します。

```
$ oc adm must-gather --image=$(oc get csv compliance-operator.v1.6.0 -
o=jsonpath='{.spec.relatedImages[?(@.name=="must-gather")].image}')
```

5.3.4. 関連情報

- must-gather ツールについて
- Product Compliance

5.4. COMPLIANCE OPERATOR の概念

5.4.1. Compliance Operator について

Compliance Operator を使用すると、OpenShift Container Platform 管理者はクラスターの必要なコンプライアンス状態を記述し、存在するギャップやそれらを修復する方法に関する概要を提供します。Compliance Operator は、OpenShift Container Platform の Kubernetes API リソースと、クラスターを実行するノードの両方のコンプライアンスを評価します。Compliance Operator は、NIST 認定ツールである OpenSCAP を使用して、コンテンツが提供するセキュリティーポリシーをスキャンし、これを適用します。

重要

Compliance Operator は Red Hat Enterprise Linux CoreOS (RHCOS) デプロイメントでのみ利用できます。

5.4.1.1. Compliance Operator のプロファイル

Compliance Operator のインストールの一部として利用可能なプロファイルは複数あります。**oc get** コマンドを使用して、使用可能なプロファイル、プロファイルの詳細、および特定のルールを表示できます。

- 利用可能なプロファイルを表示します。

```
$ oc get profile.compliance -n openshift-compliance
```

出力例

```
NAME                  AGE    VERSION
ocp4-cis              3h49m  1.5.0
ocp4-cis-1-4          3h49m  1.4.0
ocp4-cis-1-5          3h49m  1.5.0
ocp4-cis-node         3h49m  1.5.0
ocp4-cis-node-1-4     3h49m  1.4.0
ocp4-cis-node-1-5     3h49m  1.5.0
ocp4-e8               3h49m
ocp4-high             3h49m  Revision 4
ocp4-high-node        3h49m  Revision 4
ocp4-high-node-rev-4  3h49m  Revision 4
ocp4-high-rev-4       3h49m  Revision 4
ocp4-moderate         3h49m  Revision 4
ocp4-moderate-node    3h49m  Revision 4
ocp4-moderate-node-rev-4  3h49m  Revision 4
ocp4-moderate-rev-4   3h49m  Revision 4
ocp4-nerc-cip         3h49m
ocp4-nerc-cip-node    3h49m
ocp4-pci-dss          3h49m  3.2.1
ocp4-pci-dss-3-2      3h49m  3.2.1
ocp4-pci-dss-4-0      3h49m  4.0.0
ocp4-pci-dss-node     3h49m  3.2.1
ocp4-pci-dss-node-3-2  3h49m  3.2.1
ocp4-pci-dss-node-4-0  3h49m  4.0.0
ocp4-stig             3h49m  V2R1
ocp4-stig-node        3h49m  V2R1
ocp4-stig-node-v1r1   3h49m  V1R1
ocp4-stig-node-v2r1   3h49m  V2R1
ocp4-stig-v1r1        3h49m  V1R1
ocp4-stig-v2r1        3h49m  V2R1
rhcos4-e8             3h49m
rhcos4-high           3h49m  Revision 4
rhcos4-high-rev-4     3h49m  Revision 4
rhcos4-moderate       3h49m  Revision 4
rhcos4-moderate-rev-4  3h49m  Revision 4
rhcos4-nerc-cip       3h49m
rhcos4-stig           3h49m  V2R1
rhcos4-stig-v1r1      3h49m  V1R1
rhcos4-stig-v2r1      3h49m  V2R1
```

これらのプロファイルは、複数の異なるコンプライアンスベンチマークを表します。各プロファイルには、適用先の製品名がプロファイル名の接頭辞として追加されます。**ocp4-e8** は Essential 8 ベンチマークを OpenShift Container Platform 製品に適用し、**rhcos4-e8** は Essential 8 ベンチマークを Red Hat Enterprise Linux CoreOS (RHCOS) 製品に適用します。

- 以下のコマンドを実行して、**rhcos4-e8** プロファイルの詳細を表示します。

```
$ oc get -n openshift-compliance -oyaml profiles.compliance rhcos4-e8
```

例5.1 出力例


```yaml
apiVersion: compliance.openshift.io/v1alpha1
description: 'This profile contains configuration checks for Red Hat Enterprise Linux
  CoreOS that align to the Australian Cyber Security Centre (ACSC) Essential Eight.
  A copy of the Essential Eight in Linux Environments guide can be found at the ACSC
  website: https://www.cyber.gov.au/acsc/view-all-content/publications/hardening-linux-workstations-and-servers'
id: xccdf_org.ssgproject.content_profile_e8
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/product: redhat_enterprise_linux_coreos_4
    compliance.openshift.io/product-type: Node
  creationTimestamp: "2022-10-19T12:06:49Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-e8
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "43699"
  uid: 86353f70-28f7-40b4-bf0e-6289ec33675b
rules:
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
- rhcos4-audit-rules-execution-chcon
- rhcos4-audit-rules-execution-restorecon
- rhcos4-audit-rules-execution-semanage
- rhcos4-audit-rules-execution-setfiles
- rhcos4-audit-rules-execution-setsebool
- rhcos4-audit-rules-execution-seunshare
- rhcos4-audit-rules-kernel-module-loading-delete
- rhcos4-audit-rules-kernel-module-loading-finit
- rhcos4-audit-rules-kernel-module-loading-init
- rhcos4-audit-rules-login-events
- rhcos4-audit-rules-login-events-faillock
- rhcos4-audit-rules-login-events-lastlog
- rhcos4-audit-rules-login-events-tallylog
- rhcos4-audit-rules-networkconfig-modification
- rhcos4-audit-rules-sysadmin-actions
- rhcos4-audit-rules-time-adjtimex
- rhcos4-audit-rules-time-clock-settime
```

```
- rhcos4-audit-rules-time-settimeofday
- rhcos4-audit-rules-time-stime
- rhcos4-audit-rules-time-watch-localtime
- rhcos4-audit-rules-usergroup-modification
- rhcos4-auditd-data-retention-flush
- rhcos4-auditd-freq
- rhcos4-auditd-local-events
- rhcos4-auditd-log-format
- rhcos4-auditd-name-format
- rhcos4-auditd-write-logs
- rhcos4-configure-crypto-policy
- rhcos4-configure-ssh-crypto-policy
- rhcos4-no-empty-passwords
- rhcos4-selinux-policytype
- rhcos4-selinux-state
- rhcos4-service-auditd-enabled
- rhcos4-sshd-disable-empty-passwords
- rhcos4-sshd-disable-gssapi-auth
- rhcos4-sshd-disable-rhosts
- rhcos4-sshd-disable-root-login
- rhcos4-sshd-disable-user-known-hosts
- rhcos4-sshd-do-not-permit-user-env
- rhcos4-sshd-enable-strictmodes
- rhcos4-sshd-print-last-log
- rhcos4-sshd-set-loglevel-info
- rhcos4-sysctl-kernel-dmesg-restrict
- rhcos4-sysctl-kernel-kptr-restrict
- rhcos4-sysctl-kernel-randomize-va-space
- rhcos4-sysctl-kernel-unprivileged-bpf-disabled
- rhcos4-sysctl-kernel-yama-ptrace-scope
- rhcos4-sysctl-net-core-bpf-jit-harden
title: Australian Cyber Security Centre (ACSC) Essential Eight
```

- 次のコマンドを実行して、**rhcos4-audit-rules-login-events** ルールの詳細を表示します。

```
$ oc get -n openshift-compliance -oyaml rules rhcos4-audit-rules-login-events
```

例5.2 出力例

```
apiVersion: compliance.openshift.io/v1alpha1
checkType: Node
description: |-
  The audit system already collects login information for all users and root. If the auditd
daemon is configured to use the augenrules program to read audit rules during daemon
startup (the default), add the following lines to a file with suffix.rules in the directory
/etc/audit/rules.d in order to watch for attempted manual edits of files involved in storing
logon events:

  -w /var/log/tallylog -p wa -k logins
  -w /var/run/faillock -p wa -k logins
  -w /var/log/lastlog -p wa -k logins

  If the auditd daemon is configured to use the auditctl utility to read audit rules during
daemon startup, add the following lines to /etc/audit/audit.rules file in order to watch for
```

```
unattempted manual edits of files involved in storing logon events:

  -w /var/log/tallylog -p wa -k logins
  -w /var/run/faillock -p wa -k logins
  -w /var/log/lastlog -p wa -k logins
id: xccdf_org.ssgproject.content_rule_audit_rules_login_events
kind: Rule
metadata:
  annotations:
    compliance.openshift.io/image-digest: pb-rhcos4hrdkm
    compliance.openshift.io/rule: audit-rules-login-events
    control.compliance.openshift.io/NIST-800-53: AU-2(d);AU-12(c);AC-6(9);CM-6(a)
    control.compliance.openshift.io/PCI-DSS: Req-10.2.3
    policies.open-cluster-management.io/controls: AU-2(d),AU-12(c),AC-6(9),CM-
6(a),Req-10.2.3
    policies.open-cluster-management.io/standards: NIST-800-53,PCI-DSS
  creationTimestamp: "2022-10-19T12:07:08Z"
  generation: 1
  labels:
    compliance.openshift.io/profile-bundle: rhcos4
  name: rhcos4-audit-rules-login-events
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: rhcos4
    uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
  resourceVersion: "44819"
  uid: 75872f1f-3c93-40ca-a69d-44e5438824a4
rationale: Manual editing of these files may indicate nefarious activity, such as
  an attacker attempting to remove evidence of an intrusion.
severity: medium
title: Record Attempts to Alter Logon and Logout Events
warning: Manual editing of these files may indicate nefarious activity, such as an
  attacker attempting to remove evidence of an intrusion.
```


5.4.1.1.1. Compliance Operator のプロファイルタイプ

Compliance Operator のルールはプロファイルとしてまとめられます。プロファイルは、OpenShift Container Platform のプラットフォームまたはノードを対象としています。一部のベンチマークには、**rhcos4** ノードプロファイルが含まれています。

プラットフォーム

: プラットフォームプロファイルは、OpenShift Container Platform クラスターのコンポーネントを評価します。たとえば、プラットフォームレベルのルールで、APIServer 設定で強力な暗号化アルゴリズムが使用されているかどうかを確認できます。

ノード

: ノードプロファイルは、各ホストの OpenShift または RHCOS 設定を評価します。**ocp4** ノードプロファイルと **rhcos4** ノードプロファイルの 2 つのノードプロファイルを使用できます。**ocp4** ノードプロファイルは、各ホストの OpenShift 設定を評価します。たとえば、**kubeconfig** ファイルにコン

プライアンス標準に適合した正しい権限があるかどうかを確認できます。**rhcos4** ノードプロファイルは、各ホストの Red Hat Enterprise Linux CoreOS (RHCOS) 設定を評価します。たとえば、SSHD サービスがパスワードログインを無効にするように設定されているかどうかを確認できます。

重要

ノードプロファイルとプラットフォームプロファイル (PCI-DSS など) を含むベンチマークの場合、OpenShift Container Platform 環境で両方のプロファイルを実行する必要があります。

ocp4 プラットフォーム、**ocp4** ノード、および **rhcos4** ノードプロファイルを含むベンチマーク (FedRAMP High など) の場合、OpenShift Container Platform 環境で 3 つのプロファイルをすべて実行する必要があります。

注記

多数のノードを持つクラスターでは、**ocp4** ノードと **rhcos4** ノードの両方のスキャンが完了するまでに長い時間がかかる可能性があります。

5.4.2. カスタムリソース定義を理解する

OpenShift Container Platform の Compliance Operator は、コンプライアンススキャンを実行するためのいくつかのカスタムリソース定義 (CRD) を提供します。コンプライアンススキャンを実行するには、ComplianceAsCode コミュニティープロジェクトから派生した事前定義されたセキュリティーポリシーを利用します。Compliance Operator はこれらのセキュリティーポリシーを CRD に変換します。この CRD を使用してコンプライアンススキャンを実行し、見つかった問題に対する修復を行うことができます。

5.4.2.1. CRD ワークフロー

CRD は、コンプライアンススキャンを完了するための次のワークフローを提供します。

1. コンプライアンススキャン要件を定義する
2. コンプライアンススキャン設定を設定する
3. コンプライアンススキャン設定を使用してコンプライアンス要件を処理する
4. コンプライアンススキャンをモニターする
5. コンプライアンススキャンの結果を確認する

5.4.2.2. コンプライアンススキャン要件の定義

デフォルトでは、Compliance Operator CRD には **ProfileBundle** オブジェクトと **Profile** オブジェクトが含まれており、これらのオブジェクトでコンプライアンススキャン要件のルールを定義および設定できます。**TailoredProfile** オブジェクトを使用して、デフォルトのプロファイルをカスタマイズすることもできます。

5.4.2.2.1. ProfileBundle オブジェクト

Compliance Operator のインストール時に、すぐに実行できる **ProfileBundle** オブジェクトが含まれます。Compliance Operator は **ProfileBundle** オブジェクトを解析し、バンドル内の各プロファイルに対して **Profile** オブジェクトを作成します。また、**Profile** オブジェクトによって使用される **Rule** オブ

ジェクトと **Variable** オブジェクトも解析します。

ProfileBundle オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
  name: <profile bundle name>
  namespace: openshift-compliance
status:
  dataStreamStatus: VALID 1
```

1. Compliance Operator がコンテンツファイルを解析できたかどうかを示します。

> **注記**
>
> **contentFile** が失敗すると、発生したエラーの詳細を提供する **errorMessage** 属性が表示されます。

トラブルシューティング

無効なイメージから既知のコンテンツイメージにロールバックすると、**ProfileBundle** オブジェクトは応答を停止し、**PENDING** 状態を表示します。回避策として、前のイメージとは異なるイメージに移動できます。または、**ProfileBundle** オブジェクトを削除して再作成し、作業状態に戻すこともできます。

5.4.2.2.2. プロファイルオブジェクト

Profile オブジェクトは、特定のコンプライアンス標準を評価できるルールと変数を定義します。XCCDF 識別子や **Node** または **Platform** タイプのプロファイルチェックなど、OpenSCAP プロファイルに関する解析済みの詳細が含まれています。**Profile** オブジェクトを直接使用することも、**TailorProfile** オブジェクトを使用してさらにカスタマイズすることもできます。

> **注記**
>
> **Profile** オブジェクトは単一の **ProfileBundle** オブジェクトから派生しているため、手動で作成または変更することはできません。通常、1 つの **ProfileBundle** オブジェクトに複数の **Profile** オブジェクトを含めることができます。

Profile オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
description: <description of the profile>
id: xccdf_org.ssgproject.content_profile_moderate 1
kind: Profile
metadata:
  annotations:
    compliance.openshift.io/product: <product name>
    compliance.openshift.io/product-type: Node 2
  creationTimestamp: "YYYY-MM-DDTMM:HH:SSZ"
  generation: 1
  labels:
```

```
    compliance.openshift.io/profile-bundle: <profile bundle name>
  name: rhcos4-moderate
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ProfileBundle
    name: <profile bundle name>
    uid: <uid string>
  resourceVersion: "<version number>"
  selfLink: /apis/compliance.openshift.io/v1alpha1/namespaces/openshift-compliance/profiles/rhcos4-
moderate
  uid: <uid string>
rules: (3)
- rhcos4-account-disable-post-pw-expiration
- rhcos4-accounts-no-uid-except-zero
- rhcos4-audit-rules-dac-modification-chmod
- rhcos4-audit-rules-dac-modification-chown
title: <title of the profile>
```

1. プロファイルの XCCDF 名を指定します。スキャンのプロファイル属性の値として **ComplianceScan** オブジェクトを定義する場合は、この識別子を使用します。
2. **Node** または **Platform** のいずれかを指定します。ノードプロファイルはクラスターノードをスキャンし、プラットフォームプロファイルは Kubernetes プラットフォームをスキャンします。
3. プロファイルのルールのリストを指定します。各ルールは 1 つのチェックに対応します。

5.4.2.2.3. ルールオブジェクト

プロファイルを形成する **Rule** オブジェクトも、オブジェクトとして公開されます。**Rule** オブジェクトを使用して、コンプライアンスチェック要件を定義し、それを修正する方法を指定します。

Rule オブジェクトの例

```
    apiVersion: compliance.openshift.io/v1alpha1
    checkType: Platform (1)
    description: <description of the rule>
    id: xccdf_org.ssgproject.content_rule_configure_network_policies_namespaces (2)
    instructions: <manual instructions for the scan>
    kind: Rule
    metadata:
      annotations:
        compliance.openshift.io/rule: configure-network-policies-namespaces
        control.compliance.openshift.io/CIS-OCP: 5.3.2
        control.compliance.openshift.io/NERC-CIP: CIP-003-3 R4;CIP-003-3 R4.2;CIP-003-3
          R5;CIP-003-3 R6;CIP-004-3 R2.2.4;CIP-004-3 R3;CIP-007-3 R2;CIP-007-3 R2.1;CIP-007-3
          R2.2;CIP-007-3 R2.3;CIP-007-3 R5.1;CIP-007-3 R6.1
        control.compliance.openshift.io/NIST-800-53: AC-4;AC-4(21);CA-3(5);CM-6;CM-6(1);CM-7;CM-
7(1);SC-7;SC-7(3);SC-7(5);SC-7(8);SC-7(12);SC-7(13);SC-7(18)
      labels:
        compliance.openshift.io/profile-bundle: ocp4
      name: ocp4-configure-network-policies-namespaces
```

```
  namespace: openshift-compliance
rationale: <description of why this rule is checked>
severity: high 3
title: <summary of the rule>
```

1. このルールが実行するチェックのタイプを指定します。**Node** プロファイルはクラスターノードをスキャンし、**Platform** プロファイルは Kubernetes プラットフォームをスキャンします。空の値は、自動チェックがないことを示します。
2. データストリームから直接解析されるルールの XCCDF 名を指定します。
3. 失敗したときのルールの重大度を指定します。

> **注記**
>
> **Rule** オブジェクトは、関連付けられた **ProfileBundle** オブジェクトを簡単に識別できるように適切なラベルを取得します。**ProfileBundle** は、このオブジェクトの **OwnerReferences** でも指定されます。

5.4.2.2.4. TailoredProfile オブジェクト

TailoredProfile オブジェクトを使用して、組織の要件に基づいてデフォルトの **Profile** オブジェクトを変更します。ルールを有効または無効にしたり、変数値を設定したり、カスタマイズの正当性を示したりすることができます。検証後、**TailoredProfile** オブジェクトは **ConfigMap** を作成します。これは、**ComplianceScan** オブジェクトから参照できます。

> **ヒント**
>
> **ScanSettingBinding** オブジェクトで参照することにより、**TailoredProfile** オブジェクトを使用できます。**ScanSettingBinding** の詳細は、ScanSettingBinding オブジェクトを参照してください。

TailoredProfile オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: rhcos4-with-usb
spec:
  extends: rhcos4-moderate 1
  title: <title of the tailored profile>
  disableRules:
    - name: <name of a rule object to be disabled>
      rationale: <description of why this rule is checked>
status:
  id: xccdf_compliance.openshift.io_profile_rhcos4-with-usb 2
  outputRef:
    name: rhcos4-with-usb-tp 3
    namespace: openshift-compliance
  state: READY 4
```

1. これは任意です。**TailoredProfile** がビルドされる **Profile** オブジェクトの名前。値が設定されていない場合は、**enableRules** リストから新しいプロファイルが作成されます。

2 調整されたプロファイルの XCCDF 名を指定します。

3 **ConfigMap** 名を指定します。これは、**ComplianceScan** の **tailoringConfigMap.name** 属性の値として使用できます。

4 **READY**、**PENDING**、**FAILURE** などのオブジェクトの状態を表示します。オブジェクトの状態が **ERROR** の場合、属性 **status.errorMessage** が失敗の理由を提供します。

TailoredProfile オブジェクトを使用すると、**TailoredProfile** コンストラクトを使用して新しい **Profile** オブジェクトを作成できます。新しい **Profile** を作成するには、次の設定パラメーターを設定します。

- 適切なタイトル
- **extends** は空でなければなりません
- **TailoredProfile** オブジェクトのスキャンタイプアノテーション:

```
compliance.openshift.io/product-type: Platform/Node
```

> **注記**
>
> **product-type** のアノテーションを設定していない場合、Compliance Operator はデフォルトで **Platform** スキャンタイプになります。**TailoredProfile** オブジェクトの名前に **-node** 接尾辞を追加すると、**node** スキャンタイプになります。

5.4.2.3. コンプライアンススキャン設定の設定

コンプライアンススキャンの要件を定義した後、スキャンのタイプ、スキャンの発生、およびスキャンの場所を指定することにより、コンプライアンススキャンを設定できます。そのために、Compliance Operator は **ScanSetting** オブジェクトを提供します。

5.4.2.3.1. ScanSetting オブジェクト

ScanSetting オブジェクトを使用して、スキャンを実行するための運用ポリシーを定義および再利用します。デフォルトでは、コンプライアンスオペレータは次の **ScanSetting** オブジェクトを作成します。

- **default** - 1Gi Persistent Volume (PV) を使用して、マスターノードとワーカーノードの両方で毎日午前 1 時にスキャンを実行し、最後の 3 つの結果を保持します。修復は自動的に適用も更新もされません。
- **default-auto-apply** - 1Gi Persistent Volume (PV) を使用して、コントロールプレーンとワーカーノードの両方で毎日午前 1 時にスキャンを実行し、最後の 3 つの結果を保持します。**autoApplyRemediations** と **autoUpdateRemediations** の両方が true に設定されています。

ScanSetting オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
autoApplyRemediations: true (1)
autoUpdateRemediations: true (2)
kind: ScanSetting
maxRetryOnTimeout: 3
```

```
metadata:
  creationTimestamp: "2022-10-18T20:21:00Z"
  generation: 1
  name: default-auto-apply
  namespace: openshift-compliance
  resourceVersion: "38840"
  uid: 8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3 (3)
  size: 1Gi (4)
  tolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
    operator: Exists
  - effect: NoExecute
    key: node.kubernetes.io/not-ready
    operator: Exists
    tolerationSeconds: 300
  - effect: NoExecute
    key: node.kubernetes.io/unreachable
    operator: Exists
    tolerationSeconds: 300
  - effect: NoSchedule
    key: node.kubernetes.io/memory-pressure
    operator: Exists
roles: (5)
- master
- worker
scanTolerations:
- operator: Exists
schedule: 0 1 * * * (6)
showNotApplicable: false
strictNodeScan: true
timeout: 30m
```

1. 自動修復を有効にするには、**true** に設定します。自動修復を無効にするには、**false** に設定します。

2. コンテンツ更新の自動修復を有効にするには、**true** に設定します。コンテンツ更新の自動修復を無効にするには、**false** に設定します。

3. 生の結果形式で保存されたスキャンの数を指定します。デフォルト値は **3** です。古い結果がローテーションされると、管理者はローテーションが発生する前に結果を別の場所に保存する必要があります。

4. 生の結果を保存するためにスキャン用に作成する必要があるストレージサイズを指定します。デフォルト値は **1Gi** です。

6. スキャンを実行する頻度を cron 形式で指定します。

> **注記**
>
> ローテーションポリシーを無効にするには、値を **0** に設定します。

5 **node-role.kubernetes.io** ラベル値を指定して、**Node** タイプのスキャンをスケジュールします。この値は、**MachineConfigPool** の名前と一致する必要があります。

5.4.2.4. コンプライアンススキャン設定を使用したコンプライアンススキャン要件の処理

コンプライアンススキャンの要件を定義し、スキャンを実行するための設定を行うと、Compliance Operator が **ScanSettingBinding** オブジェクトを使用してそれを処理します。

5.4.2.4.1. ScanSettingBinding オブジェクト

ScanSettingBinding オブジェクトを使用して、**Profile** または **TailoredProfile** オブジェクトを参照してコンプライアンス要件を指定します。次に、スキャンの操作上の制約を提供する **ScanSetting** オブジェクトにリンクされます。次に、Compliance Operator は、**ScanSetting** オブジェクトと **ScanSettingBinding** オブジェクトに基づいて **ComplianceSuite** オブジェクトを生成します。

ScanSettingBinding オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: <name of the scan>
profiles: 1
  # Node checks
  - name: rhcos4-with-usb
    kind: TailoredProfile
    apiGroup: compliance.openshift.io/v1alpha1
  # Cluster checks
  - name: ocp4-moderate
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef: 2
  name: my-companys-constraints
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

1 **Profile** または **TailoredProfile** オブジェクトの詳細を指定して、環境をスキャンします。

2 スケジュールやストレージサイズなどの運用上の制約を指定します。

ScanSetting オブジェクトと **ScanSettingBinding** オブジェクトを作成すると、コンプライアンススイートが作成されます。コンプライアンススイートのリストを取得するには、次のコマンドを実行します。

```
$ oc get compliancesuites
```

重要

ScanSettingBinding を削除すると、コンプライアンススイートも削除されます。

5.4.2.5. コンプライアンススキャンの追跡

コンプライアンススイートの作成後、**ComplianceSuite** オブジェクトを使用して、デプロイされたスキャンのステータスをモニターできます。

5.4.2.5.1. ComplianceSuite オブジェクト

ComplianceSuite オブジェクトは、スキャンの状態を追跡するのに役立ちます。スキャンと全体的な結果を作成するための生の設定が含まれています。

Node タイプのスキャンの場合、問題の修正が含まれているため、スキャンを **MachineConfigPool** にマップする必要があります。ラベルを指定する場合は、それがプールに直接適用されることを確認してください。

ComplianceSuite オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: <name of the scan>
spec:
  autoApplyRemediations: false 1
  schedule: "0 1 * * *" 2
  scans: 3
    - name: workers-scan
      scanType: Node
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc...
      rule: "xccdf_org.ssgproject.content_rule_no_netrc_files"
      nodeSelector:
        node-role.kubernetes.io/worker: ""
status:
  Phase: DONE 4
  Result: NON-COMPLIANT 5
  scanStatuses:
  - name: workers-scan
    phase: DONE
    result: NON-COMPLIANT
```

1 自動修復を有効にするには、**true** に設定します。自動修復を無効にするには、**false** に設定します。

2 スキャンを実行する頻度を cron 形式で指定します。

3 クラスターで実行するスキャン仕様のリストを指定します。

4 スキャンの進行状況を示します。

5 スイートの全体的な判断を示します。

バックグラウンドのスイートは、**scans** パラメーターに基づいて **ComplianceScan** オブジェクトを作成します。プログラムで **ComplianceSuites** イベントを取得できます。スイートのイベントを取得するには、次のコマンドを実行します。

```
$ oc get events --field-selector involvedObject.kind=ComplianceSuite,involvedObject.name=<name of the suite>
```

重要

手動で **ComplianceSuite** を定義すると、XCCDF 属性が含まれているため、エラーが発生する可能性があります。

5.4.2.5.2. 高度な ComplianceScan オブジェクト

Compliance Operator は、デバッグや既存のツールとの統合を行うための上級ユーザー向けのオプションを備えています。**ComplianceScan** オブジェクトを直接作成しないことを推奨しますが、代わりに、**ComplianceSuite** オブジェクトを使用してオブジェクトを管理できます。

高度な ComplianceScan オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceScan
metadata:
  name: <name of the scan>
spec:
  scanType: Node 1
  profile: xccdf_org.ssgproject.content_profile_moderate 2
  content: ssg-ocp4-ds.xml
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc...
3
  rule: "xccdf_org.ssgproject.content_rule_no_netrc_files" 4
  nodeSelector: 5
    node-role.kubernetes.io/worker: ""
status:
  phase: DONE 6
  result: NON-COMPLIANT 7
```

1. **Node** または **Platform** のいずれかを指定します。ノードプロファイルはクラスターノードをスキャンし、プラットフォームプロファイルは Kubernetes プラットフォームをスキャンします。
2. 実行するプロファイルの XCCDF 識別子を指定します。
3. プロファイルファイルをカプセル化するコンテナーイメージを指定します。
4. これはオプションです。単一のルールを実行するスキャンを指定します。このルールは XCCDF ID で識別され、指定されたプロファイルに属している必要があります。

注記

rule パラメーターをスキップすると、指定されたプロファイルで使用可能なすべてのルールに対してスキャンが実行されます。

5 OpenShift Container Platform を使用していて、修復を生成したい場合は、nodeSelector ラベルが **MachineConfigPool** ラベルと一致する必要があります。

> **注記**
>
> **nodeSelector** パラメーターを指定しないか、**MachineConfig** ラベルと一致しない場合でも、スキャンは実行されますが、修復は作成されません。

6 スキャンの現在のフェーズを示します。

7 スキャンの判定を示します。

> **重要**
>
> **ComplianceSuite** オブジェクトを削除すると、関連するすべてのスキャンが削除されます。

スキャンが完了すると、**ComplianceCheckResult** オブジェクトのカスタムリソースとして結果が生成されます。ただし、生の結果は ARF 形式で入手できます。これらの結果は、スキャンの名前に関連付けられた永続ボリュームクレーム (PVC) を持つ永続ボリューム (PV) に保存されます。プログラムで **ComplianceScans** イベントを取得できます。スイートのイベントを生成するには、次のコマンドを実行します。

```
oc get events --field-selector involvedObject.kind=ComplianceScan,involvedObject.name=<name of the suite>
```

5.4.2.6. コンプライアンス結果の表示

コンプライアンススイートが **DONE** フェーズに達すると、スキャン結果と可能な修正を表示できます。

5.4.2.6.1. ComplianceCheckResult オブジェクト

特定のプロファイルでスキャンを実行すると、プロファイル内のいくつかのルールが検証されます。これらのルールごとに、**ComplianceCheckResult** オブジェクトが作成され、特定のルールのクラスターの状態が提供されます。

ComplianceCheckResult オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceCheckResult
metadata:
  labels:
    compliance.openshift.io/check-severity: medium
    compliance.openshift.io/check-status: FAIL
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
  name: workers-scan-no-direct-root-logins
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
```

```
    controller: true
    kind: ComplianceScan
    name: workers-scan
description: <description of scan check>
instructions: <manual instructions for the scan>
id: xccdf_org.ssgproject.content_rule_no_direct_root_logins
severity: medium 1
status: FAIL 2
```

1 スキャンチェックの重大度を説明します。

2 チェックの結果を説明します。以下の値を使用できます。

- PASS: チェックは成功しました。
- FAIL: チェックに失敗しました。
- INFO: チェックは成功し、エラーと見なされるほど深刻ではないものが見つかりました。
- MANUAL: チェックではステータスを自動的に評価できないため、手動チェックが必要です。
- INCONSISTENT: ノードが異なれば、報告される結果も異なります。
- ERROR: チェックは正常に実行されましたが、完了できませんでした。
- NOTAPPLICABLE: 該当しないため、チェックは実行されませんでした。

スイートからすべてのチェック結果を取得するには、次のコマンドを実行します。

```
oc get compliancecheckresults \
-l compliance.openshift.io/suite=workers-compliancesuite
```

5.4.2.6.2. ComplianceRemediation オブジェクト

チェック項目によっては、データストリームで指定された修正方法が存在する場合があります。一方、Kubernetes の修正が利用可能な場合、Compliance Operator によって **ComplianceRemediation** オブジェクトが作成されます。

ComplianceRemediation オブジェクトの例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  labels:
    compliance.openshift.io/suite: example-compliancesuite
    compliance.openshift.io/scan-name: workers-scan
    machineconfiguration.openshift.io/role: worker
  name: workers-scan-disable-users-coredumps
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
```

```
    kind: ComplianceCheckResult
    name: workers-scan-disable-users-coredumps
    uid: <UID>
spec:
  apply: false 1
  object:
    current: 2
       apiVersion: machineconfiguration.openshift.io/v1
       kind: MachineConfig
       spec:
         config:
           ignition:
             version: 2.2.0
           storage:
             files:
             - contents:
                 source: data:,%2A%20%20%20%20%20hard%20%20%20core%20%20%20%20%200
               filesystem: root
               mode: 420
               path: /etc/security/limits.d/75-disable_users_coredumps.conf
    outdated: {} 3
```

1. **true** は、修復が適用されたことを示します。**false** は、修復が適用されなかったことを示します。
2. 修復の定義が含まれています。
3. 以前のバージョンのコンテンツから以前に解析された修復を示します。Compliance Operator は、管理者が新しい修復を適用する前に確認できるように、古いオブジェクトを引き続き保持します。

スイートからすべての修復を取得するには、次のコマンドを実行します。

```
oc get complianceremediations \
-l compliance.openshift.io/suite=workers-compliancesuite
```

自動的に修正できるすべての失敗したチェックをリスト表示するには、次のコマンドを実行します。

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),compliance.openshift.io/automated-remediation'
```

手動で修正できるすべての失敗したチェックをリスト表示するには、次のコマンドを実行します。

```
oc get compliancecheckresults \
-l 'compliance.openshift.io/check-status in (FAIL),!compliance.openshift.io/automated-remediation'
```

5.5. COMPLIANCE OPERATOR の管理

5.5.1. Compliance Operator のインストール

Compliance Operator を使用する前に、これがクラスターにデプロイされていることを確認する必要があります。

重要

この Operator を適切に機能させるには、すべてのクラスターノードのリリースバージョンが同じである必要があります。たとえば、RHCOS を実行しているノードの場合、すべてのノードの RHCOS バージョンが同じである必要があります。

重要

Compliance Operator は、OpenShift Dedicated、Red Hat OpenShift Service on AWS Classic、Azure Red Hat OpenShift などのマネージドプラットフォームで誤った結果を報告する場合があります。詳細は、ナレッジベースの記事 Compliance Operator reports incorrect results on Managed Services を参照してください。

重要

Compliance Operator をデプロイする前に、raw の結果出力を保存するためにクラスター内に永続ストレージを定義する必要があります。詳細は、永続ストレージの概要 および デフォルトのストレージクラスの管理 を参照してください。

5.5.1.1. Web コンソールを使用した Compliance Operator のインストール

前提条件

- **admin** 権限がある。
- **StorageClass** リソースが設定されている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. Compliance Operator を検索し、**Install** をクリックします。
3. **Installation mode** および **namespace** のデフォルトの選択を維持し、Operator が **openshift-compliance** namespace にインストールされていることを確認します。
4. **Install** をクリックします。

検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators → Installed Operators** ページに移動します。
2. Compliance Operator が **openshift-compliance** namespace にインストールされ、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators → Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads → Pods** ページに移動し、**openshift-compliance** プロジェクトの Pod で問題を報告しているログの有無を確認します。

重要

restricted な Security Context Constraints (SCC) が **system:authenticated** グループを含むように変更されているか、**requiredDropCapabilities** を追加している場合、権限の問題により Compliance Operator が正しく機能しない可能性があります。

Compliance Operator スキャナー Pod サービスアカウント用のカスタム SCC を作成できます。詳細は Compliance Operator のカスタム SCC の作成 を参照してください。

5.5.1.2. CLI を使用した Compliance Operator のインストール

前提条件

- **admin** 権限がある。
- **StorageClass** リソースが設定されている。

手順

1. **Namespace** オブジェクトを定義します。

   namespace-object.yaml の例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged 1
  name: openshift-compliance
```

   1 OpenShift Container Platform 4.16 では、Pod セキュリティーラベルを namespace レベルで **privileged** に設定する必要があります。

2. **Namespace** オブジェクトを作成します。

```
$ oc create -f namespace-object.yaml
```

3. **OperatorGroup** オブジェクトを定義します。

   operator-group-object.yaml の例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  targetNamespaces:
  - openshift-compliance
```

4. **OperatorGroup** オブジェクトを作成します。

```
$ oc create -f operator-group-object.yaml
```

5. **Subscription** オブジェクトを定義します。

subscription-object.yaml の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

6. **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription-object.yaml
```

注記

グローバルスケジューラー機能を設定し、**defaultNodeSelector** を有効にする場合は、namespace を手動で作成し、**openshift-compliance** namespace または Compliance Operator がインストールされている namespace のアノテーションを **openshift.io/node-selector: “”** で更新する必要があります。これにより、デフォルトのノードセレクターが削除され、デプロイメントの失敗を防ぐことができます。

検証

1. CSV ファイルを確認して、インストールが正常に完了したことを確認します。

```
$ oc get csv -n openshift-compliance
```

2. Compliance Operator が稼働していることを確認します。

```
$ oc get deploy -n openshift-compliance
```

5.5.1.3. ROSA Hosted Control Plane (HCP) への Compliance Operator のインストール

Compliance Operator 1.5.0 リリース以降、Operator は Hosted Control Plane を使用して AWS 上の Red Hat OpenShift Service に対してテストされています。

Red Hat OpenShift Service on AWS Hosted Control Plane クラスターは、Red Hat によって管理されるコントロールプレーンへのアクセスが制限されています。デフォルトでは、Compliance Operator は **master** ノードプール内のノードにスケジュールしますが、これは Red Hat OpenShift Service on AWS では使用できません。これには、Operator が利用可能なノードプールでスケジュールできるように **Subscription** オブジェクトを設定する必要があります。この手順は、AWS Hosted Control Plane クラスター上の Red Hat OpenShift Service を正常にインストールするために必要です。

前提条件

- **admin** 権限がある。
- **StorageClass** リソースが設定されている。

手順

1. **Namespace** オブジェクトを定義します。

   namespace-object.yaml ファイルの例

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
     labels:
       openshift.io/cluster-monitoring: "true"
       pod-security.kubernetes.io/enforce: privileged (1)
     name: openshift-compliance
   ```

   (1) OpenShift Container Platform 4.16 では、Pod セキュリティーラベルを namespace レベルで **privileged** に設定する必要があります。

2. 次のコマンドを実行して、**Namespace** オブジェクトを作成します。

   ```
   $ oc create -f namespace-object.yaml
   ```

3. **OperatorGroup** オブジェクトを定義します。

   operator-group-object.yaml ファイルの例

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: compliance-operator
     namespace: openshift-compliance
   spec:
     targetNamespaces:
     - openshift-compliance
   ```

4. 以下のコマンドを実行して **OperatorGroup** オブジェクトを作成します。

   ```
   $ oc create -f operator-group-object.yaml
   ```

5. **Subscription** オブジェクトを定義します。

   subscription-object.yaml ファイルの例

   ```
   apiVersion: operators.coreos.com/v1alpha1
   kind: Subscription
   metadata:
     name: compliance-operator-sub
   ```

```
  namespace: openshift-compliance
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: "" 1
```

1 Operator デプロイメントを更新して、**worker** ノードにデプロイします。

6. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription-object.yaml
```

検証

1. 次のコマンドを実行してクラスターサービスバージョン (CSV) ファイルを調べ、インストールが成功したことを確認します。

```
$ oc get csv -n openshift-compliance
```

2. 次のコマンドを使用して、Compliance Operator が起動して実行されていることを確認します。

```
$ oc get deploy -n openshift-compliance
```

重要

restricted な Security Context Constraints (SCC) が **system:authenticated** グループを含むように変更されているか、**requiredDropCapabilities** を追加している場合、権限の問題により Compliance Operator が正しく機能しない可能性があります。

Compliance Operator スキャナー Pod サービスアカウント用のカスタム SCC を作成できます。詳細は Compliance Operator のカスタム SCC の作成 を参照してください。

5.5.1.4. Hypershift Hosted Control Plane への Compliance Operator のインストール

Compliance Operator は、**Subscription** ファイルを作成して OperatorHub を使用して Hosted Control Plane にインストールできます。

重要

Hosted Control Plane は、テクノロジープレビュー機能としてのみ利用できます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

前提条件

- **admin** 権限がある。

手順

1. 以下のような **Namespace** オブジェクトを定義します。

namespace-object.yaml の例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged 1
  name: openshift-compliance
```

1 OpenShift Container Platform 4.16 では、Pod セキュリティーラベルを namespace レベルで **privileged** に設定する必要があります。

2. 次のコマンドを実行して、**Namespace** オブジェクトを作成します。

```
$ oc create -f namespace-object.yaml
```

3. **OperatorGroup** オブジェクトを定義します。

operator-group-object.yaml の例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  targetNamespaces:
  - openshift-compliance
```

4. 以下のコマンドを実行して **OperatorGroup** オブジェクトを作成します。

```
$ oc create -f operator-group-object.yaml
```

■

5. **Subscription** オブジェクトを定義します。

subscription-object.yaml の例

```yaml
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: compliance-operator-sub
  namespace: openshift-compliance
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: compliance-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  config:
    nodeSelector:
      node-role.kubernetes.io/worker: ""
    env:
    - name: PLATFORM
      value: "HyperShift"
```

6. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription-object.yaml
```

検証

1. 以下のコマンドを実行して CSV ファイルを検査し、インストールが正常に完了したことを確認します。

```
$ oc get csv -n openshift-compliance
```

2. 次のコマンドを実行して、Compliance Operator が稼働していることを確認します。

```
$ oc get deploy -n openshift-compliance
```

関連情報

- Hosted Control Plane の概要

5.5.1.5. 関連情報

- Compliance Operator はネットワークが制限された環境でサポートされています。詳細は、ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。

5.5.2. Compliance Operator の更新

クラスター管理者は、OpenShift Container Platform クラスターで Compliance Operator を更新できます。

重要

OpenShift Container Platform クラスターをバージョン 4.14 に更新すると、Compliance Operator が期待どおりに動作しなくなる可能性があります。これは継続中の既知の問題が原因です。詳細は、OCPBUGS-18025 を参照してください。

5.5.2.1. Operator 更新の準備

インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャネルを指定します。更新チャネルを変更して、新しいチャネルからの更新の追跡と受信を開始できます。

サブスクリプションの更新チャネルの名前は Operators 間で異なる可能性がありますが、命名スキーム通常、特定の Operators 内の共通の規則に従います。たとえば、チャネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度 (**stable**、**fast**) に基づく可能性があります。

注記

インストールされた Operators は、現在のチャネルよりも古いチャネルに切り換えることはできません。

Red Hat Customer Portal Labs には、管理者が Operators の更新を準備するのに役立つ以下のアプリケーションが含まれています。

- Red Hat OpenShift Container Platform Operator Update Information Checker

このアプリケーションを使用して、Operator Lifecycle Manager ベースの Operator を検索し、OpenShift Container Platform の異なるバージョン間で更新チャネルごとに利用可能な Operator バージョンを確認できます。Cluster Version Operator ベースの Operator は含まれません。

5.5.2.2. Operator の更新チャネルの変更

OpenShift Container Platform Web コンソールを使用して、Operator の更新チャネルを変更できます。

ヒント

サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合、アップグレードプロセスは、選択したチャネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが **Manual** に設定されている場合は、保留中のアップグレードを手動で承認する必要があります。

前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. 更新チャネルを変更する Operator の名前をクリックします。

3. **Subscription** タブをクリックします。

4. **Update channel** の下にある更新チャネルの名前をクリックします。

5. 変更する新しい更新チャネルをクリックし、**Save** をクリックします。

6. **Automatic** 承認ストラテジーのあるサブスクリプションの場合、更新は自動的に開始します。**Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。
   Manual 承認ストラテジーのあるサブスクリプションの場合、**Subscription** タブから更新を手動で承認できます。

5.5.2.3. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。

2. 更新が保留中の Operator は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。

3. **Subscription** タブをクリックします。承認が必要な更新は、**Upgrade status** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。

4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。

5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。

6. **Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

5.5.3. Compliance Operator の管理

このセクションでは、コンプライアンスコンテンツの更新されたバージョンを使用する方法や、カスタム **ProfileBundle** オブジェクトを作成する方法など、セキュリティーコンテンツのライフサイクルを説明します。

5.5.3.1. ProfileBundle CR の例

ProfileBundle オブジェクトには、**contentImage** が含まれるコンテナーイメージの URL と、コンプライアンスコンテンツが含まれるファイルの 2 つの情報が必要です。**contentFile** パラメーターはファイルシステムのルートに相対します。以下の例のように、ビルトインの **rhcos4 ProfileBundle** オブジェクトを定義できます。

```
apiVersion: compliance.openshift.io/v1alpha1
```

```
kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"
  uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml 1
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
2
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID
```

1. コンプライアンスコンテンツが含まれるファイルの場所。
2. コンテンツイメージの場所。

重要

コンテンツイメージに使用されるベースイメージには、**coreutils** が含まれる必要があります。

5.5.3.2. セキュリティーコンテンツの更新

セキュリティーコンテンツは、**ProfileBundle** オブジェクトが参照するコンテナーイメージとして含まれます。**ProfileBundles** や、ルールまたはプロファイルなどのバンドルから解析されたカスタムリソースへの更新を正確に追跡するには、タグの代わりにダイジェストを使用してコンプライアンスコンテンツを持つコンテナーイメージを識別します。

```
$ oc -n openshift-compliance get profilebundles rhcos4 -oyaml
```

出力例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ProfileBundle
metadata:
  creationTimestamp: "2022-10-19T12:06:30Z"
  finalizers:
  - profilebundle.finalizers.compliance.openshift.io
  generation: 1
  name: rhcos4
  namespace: openshift-compliance
  resourceVersion: "46741"
```

```
  uid: 22350850-af4a-4f5c-9a42-5e7b68b82d7d
spec:
  contentFile: ssg-rhcos4-ds.xml
  contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:900e...
1
status:
  conditions:
  - lastTransitionTime: "2022-10-19T12:07:51Z"
    message: Profile bundle successfully parsed
    reason: Valid
    status: "True"
    type: Ready
  dataStreamStatus: VALID
```

1 セキュリティーコンテナーイメージ。

それぞれの **ProfileBundle** はデプロイメントでサポートされます。Compliance Operator がコンテナーイメージダイジェストが変更されたことを検知すると、デプロイメントは変更を反映し、コンテンツを再び解析するように更新されます。タグの代わりにダイジェストを使用すると、安定した予測可能なプロファイルセットを使用できます。

5.5.3.3. 関連情報

- Compliance Operator はネットワークが制限された環境でサポートされています。詳細は、ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。

5.5.4. Compliance Operator のアンインストール

OpenShift Container Platform Web コンソールまたは CLI を使用して、クラスターから OpenShift Compliance Operator を削除できます。

5.5.4.1. Web コンソールを使用した OpenShift Container Platform からの OpenShift Compliance Operator のアンインストール

Compliance Operator を削除するには、まず namespace のオブジェクトを削除する必要があります。オブジェクトが削除されたら、**openshift-compliance** プロジェクトを削除することで、Operator とその namespace を削除できます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift Compliance Operator をインストールする必要があります。

手順

OpenShift Container Platform Web コンソールを使用して Compliance Operator を削除するには、以下を行います。

1. **Operators → Installed Operators → Compliance Operator** ページに移動します。
   a. **All instances** をクリックします。

b. **All namespaces** で、 ⋮ オプションメニューをクリックし、すべての ScanSettingBinding、ComplainceSuite、ComplianceScan、および ProfileBundle オブジェクトを削除します。

2. **Administration** → **Operators** → **Installed Operators** ページに切り替えます。

3. **Compliance Operator** エントリーのオプションメニュー ⋮ をクリックして **Uninstall Operator** を選択します。

4. **Home** → **Projects** ページに切り替えます。

5. 'compliance' を検索します。

6. **openshift-compliance** プロジェクトの横にある Options メニュー ⋮ をクリックし、**Delete Project** を選択します。

   a. ダイアログボックスに **openshift-compliance** と入力して削除を確認し、**Delete** をクリックします。

5.5.4.2. CLI を使用した OpenShift Container Platform からの OpenShift Compliance Operator のアンインストール

Compliance Operator を削除するには、まず namespace のオブジェクトを削除する必要があります。オブジェクトが削除されたら、**openshift-compliance** プロジェクトを削除することで、Operator とその namespace を削除できます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- OpenShift Compliance Operator をインストールする必要があります。

手順

1. namespace のすべてのオブジェクトを削除します。

   a. **ScanSettingBinding** オブジェクトを削除します。

```
$ oc delete ssb --all -n openshift-compliance
```

   b. **ScanSetting** オブジェクトを削除します。

```
$ oc delete ss --all -n openshift-compliance
```

   c. **ComplianceSuite** オブジェクトを削除します。

```
$ oc delete suite --all -n openshift-compliance
```

d. **ComplianceScan** オブジェクトを削除します。

```
$ oc delete scan --all -n openshift-compliance
```

e. **ProfileBundle** オブジェクトを削除します。

```
$ oc delete profilebundle.compliance --all -n openshift-compliance
```

2. Subscription オブジェクトを削除します。

```
$ oc delete sub --all -n openshift-compliance
```

3. CSV オブジェクトを削除します。

```
$ oc delete csv --all -n openshift-compliance
```

4. プロジェクトを削除します。

```
$ oc delete project openshift-compliance
```

出力例

```
project.project.openshift.io "openshift-compliance" deleted
```

検証

1. namespace が削除されていることを確認します。

```
$ oc get project/openshift-compliance
```

出力例

```
Error from server (NotFound): namespaces "openshift-compliance" not found
```

5.6. COMPLIANCE OPERATOR のスキャンの管理

5.6.1. サポートされているコンプライアンスプロファイル

Compliance Operator (CO) のインストールの一部として利用可能なプロファイルは複数あります。次のプロファイルを使用すると、クラスター内のギャップを評価できます。ただし、使用するだけでは特定のプロファイルへの準拠を推測または保証できません。また、プロファイルは監査人はありません。

このようなさまざまな標準に対する準拠または認定を実現するには、Qualified Security Assessor (QSA)、Joint Authorization Board (JAB)、または業界で認められたその他の規制当局など、認定監査機関に依頼して、環境の評価を受ける必要があります。標準への準拠を実現するには、認定監査人と協力する必要があります。

すべての Red Hat 製品のコンプライアンスサポートの詳細は、Product Compliance を参照してください。

重要

Compliance Operator は、OpenShift Dedicated や Azure Red Hat OpenShift などの一部のマネージドプラットフォームで誤った結果を報告する可能性があります。詳細は、Red Hat ナレッジベースソリューション Red Hat Knowledgebase Solution #6983418 を参照してください。

5.6.1.1. コンプライアンスプロファイル

Compliance Operator は、業界標準のベンチマークを満たすプロファイルを提供します。

注記

次の表は、Compliance Operator で利用可能な最新のプロファイルを反映しています。

5.6.1.1.1. CIS コンプライアンスプロファイル

表5.1 サポートされている CIS コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-cis [1]	CIS Red Hat OpenShift Container Platform Benchmark v1.7.0	プラットフォーム	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x aarch64**	
ocp4-cis-1-7[3]	CIS Red Hat OpenShift Container Platform Benchmark v1.7.0	プラットフォーム	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x aarch64**	
ocp4-cis-node [1]	CIS Red Hat OpenShift Container Platform Benchmark v1.7.0	ノード [2]	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-cis-node-1-7[3]	CIS Red Hat OpenShift Container Platform Benchmark v1.7.0	ノード [2]	CIS Benchmarks ™ [4]	**x86_64 ppc64le s390x aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. **ocp4-cis** および **ocp4-cis-node** プロファイルは、Compliance Operator で利用可能になると、CIS ベンチマークの最新バージョンを維持します。CIS v1.7.0 などの特定のバージョンに準拠する場合は、**ocp4-cis-1-7** および **ocp4-cis-node-1-7** プロファイルを使用します。
2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。
3. 以前のすべての CIS プロファイルは CIS v1.7.0 に置き換えられました。最新のプロファイルを環境に適用することを推奨します。
4. CIS OpenShift Container Platform v4 ベンチマークを見つけるには、CIS Benchmarks にアクセスし、**Download Latest CIS Benchmark** をクリックします。そこで登録すると、ベンチマークをダウンロードできます。

5.6.1.1.2. BSI プロファイルのサポート

表5.2 サポートされている BSI コンプライアンスプロファイル

Profile	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポートされているプラットフォーム
ocp4-bsi [1]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	プラットフォーム	BSI Basic Protection Compendium	**x86_64**	

Profile	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポートされているプラットフォーム
ocp4-bsi-node [1]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	ノード [2]	BSI Basic Protection Compendium	**x86_64**	
ocp4-bsi-2022 [3]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	プラットフォーム	BSI Basic Protection Compendium	**x86_64**	
ocp4-bsi-node-2022 [3]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	ノード [2]	BSI Basic Protection Compendium	**x86_64**	
rhcos4-bsi [3]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	ノード [2]	BSI Basic Protection Compendium	**x86_64**	
ocp4-bsi-2022 [3]	BSI IT-Grundschutz (Basic Protection) Building Block SYS.1.6 and APP.4.4	ノード [2]	BSI Basic Protection Compendium	**x86_64**	

1. **ocp4-bsi** および **ocp4-bsi-node** プロファイルには、Compliance Operator で新しいバージョンが利用可能になるたびに、BSI Basic Protection Profile の最新バージョンが反映されます。BSI 2022 などの特定のバージョンに準拠する場合は、**ocp4-bsi-2022** および **ocp4-bsi-node-2022** プロファイルを使用してください。

2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

3. 2022 年版は、BSI IT-Grundschutz (Basic Protection) Compendium の最新の英語版です。最新

3. 2022 年版は、BSI IT-Grundschutz (Basic Protection) Compendium の最新の英語版です。最新のドイツ語大要 (2023 年版) では、Building Blocks SYS.1.6 and APP.4.4 に変更はありませんでした。

詳細は、BSI Quick Check を参照してください。

5.6.1.1.3. Essential Eight コンプライアンスプロファイル

表5.3 サポートされている Essential Eight コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	プラットフォーム	ACSC Hardening Linux ワークステーションおよびサーバー	**x86_64**	
rhcos4-e8	Australian Cyber Security Centre (ACSC) Essential Eight	ノード	ACSC Hardening Linux ワークステーションおよびサーバー	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

5.6.1.1.4. FedRAMP High コンプライアンスプロファイル

重要

service-sshd-disabled ルールを使用する **rhcos4-stig** などのプロファイルに自動修復を適用すると、**sshd** サービスが自動的に無効になります。この状況により、コントロールプレーンノードとコンピュートノードへの SSH アクセスがブロックされます。SSH アクセスを有効のままにするには、**TailoredProfile** オブジェクトを作成し、**disableRules** パラメーターに **rhcos4-service-sshd-disabled** ルール値を設定します。

表5.4 サポートされている FedRAMP High コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-high [1]	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - プラットフォームレベル	プラットフォーム	NIST SP-800-53 Release Search	**x86_64**	
ocp4-high-node [1]	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - ノードレベル	ノード [2]	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-high-node-rev-4	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - ノードレベル	ノード [2]	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-high-rev-4	NIST 800-53 High-Impact Baseline for Red Hat OpenShift - プラットフォームレベル	プラットフォーム	NIST SP-800-53 Release Search	**x86_64**	
rhcos4-high [1]	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	ノード	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
rhcos4-high-rev-4	NIST 800-53 High-Impact Baseline for Red Hat Enterprise Linux CoreOS	ノード	NIST SP-800-53 Release Search	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. **ocp4-high**、**ocp4-high-node**、および **rhcos4-high** プロファイルは、Compliance Operator で

利用可能になると、FedRAMP High 標準の最新バージョンを維持します。FedRAMP high R4 などの特定のバージョンに準拠する場合は、**ocp4-high-rev-4** および **ocp4-high-node-rev-4** プロファイルを使用します。

2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

5.6.1.1.5. FedRAMP Moderate コンプライアンスプロファイル

表5.5 サポートされている FedRAMP Moderate コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-moderate [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - プラットフォームレベル	プラットフォーム	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x aarch64**	
ocp4-moderate-node [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - ノードレベル	ノード [2]	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-moderate-node-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - ノードレベル	ノード [2]	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-moderate-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat OpenShift - プラットフォームレベル	プラットフォーム	NIST SP-800-53 Release Search	**x86_64 ppc64le s390x aarch64**	

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
rhcos4-moderate [1]	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	ノード	NIST SP-800-53 Release Search	**x86_64** **aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
rhcos4-moderate-rev-4	NIST 800-53 Moderate-Impact Baseline for Red Hat Enterprise Linux CoreOS	ノード	NIST SP-800-53 Release Search	**x86_64** **aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. **ocp4-moderate**、**ocp4-moderate-node**、**rhcos4-moderate** プロファイルは、Compliance Operator で利用可能になると、FedRAMP Moderate 標準の最新バージョンを維持します。FedRAMP Moderate R4 などの特定のバージョンに準拠する場合は、**ocp4-moderate-rev-4** および **ocp4-moderate-node-rev-4** プロファイルを使用します。

2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

5.6.1.1.6. NERC-CIP コンプライアンスプロファイル

表5.6 サポートされている NERC-CIP コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-nerc-cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the OpenShift Container Platform - Platform レベル	プラットフォーム	NERC CIP 標準	**x86_64**	
ocp4-nerc-cip-node	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for the OpenShift Container Platform - Node レベル	ノード [1]	NERC CIP 標準	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
rhcos4-nerc-cip	North American Electric Reliability Corporation (NERC) Critical Infrastructure Protection (CIP) cybersecurity standards profile for Red Hat Enterprise Linux CoreOS	ノード	NERC CIP 標準	**x86_64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

5.6.1.1.7. PCI-DSS コンプライアンスプロファイル

表5.7 サポートされている PCI-DSS コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-pci-dss [1]	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	プラットフォーム	PCI Security Standards ® Council Document Library	**x86_64 ppc64le aarch64**	
ocp4-pci-dss-3-2 [3]	PCI-DSS v3.2.1 Control Baseline for OpenShift Container Platform 4	プラットフォーム	PCI Security Standards ® Council Document Library	**x86_64 ppc64le s390x aarch64**	
ocp4-pci-dss-4-0	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	プラットフォーム	PCI Security Standards ® Council Document Library	**x86_64 ppc64le aarch64**	
ocp4-pci-dss-node [1]	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	ノード [2]	PCI Security Standards ® Council Document Library	**x86_64 ppc64le aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-pci-dss-node-3-2 [3]	PCI-DSS v3.2.1 Control Baseline for OpenShift Container Platform 4	ノード [2]	PCI Security Standards ® Council Document Library	**x86_64 ppc64le s390x aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-pci-dss-node-4-0	PCI-DSS v4 Control Baseline for OpenShift Container Platform 4	ノード [2]	PCI Security Standards ® Council Document Library	**x86_64 ppc64le aarch64**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. **ocp4-pci-dss** および **ocp4-pci-dss-node** プロファイルは、Compliance Operator で利用可能になると、最新バージョンの PCI-DSS 標準を維持します。PCI-DSS v3.2.1 などの特定のバージョンに準拠する場合は、**ocp4-pci-dss-3-2** および **ocp4-pci-dss-node-3-2** プロファイルを使用します。

2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

3. PCI-DSS v3.2.1 は PCI-DSS v4 に置き換えられました。最新のプロファイルを環境に適用することを推奨します。

5.6.1.1.8. STIG コンプライアンスプロファイル

重要

service-sshd-disabled ルールを使用する **rhcos4-stig** などのプロファイルに自動修復を適用すると、**sshd** サービスが自動的に無効になります。この状況により、コントロールプレーンノードとコンピュートノードへの SSH アクセスがブロックされます。SSH アクセスを有効のままにするには、**TailoredProfile** オブジェクトを作成し、**disableRules** パラメーターに **rhcos4-service-sshd-disabled** ルール値を設定します。

表5.8 サポートされている STIG コンプライアンスプロファイル

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-stig [1]	国防情報システム局セキュリティー技術実装ガイド (DISA STIG)Red Hat Openshift [3]	プラットフォーム	DISA-STIG	**x86_64 ppc64le**	
ocp4-stig-node [1]	国防情報システム局セキュリティー技術実装ガイド (DISA STIG)Red Hat Openshift [3]	ノード [2]	DISA-STIG	**x86_64 ppc64le**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
ocp4-stig-v2r3	Red Hat Openshift V2R3 向け国防情報システム局セキュリティー技術実装ガイド (DISA STIG)	プラットフォーム	DISA-STIG	**x86_64 ppc64le**	

プロファイル	プロファイルタイトル	アプリケーション	業界コンプライアンスベンチマーク	サポートされているアーキテクチャー	サポート対象のプラットフォーム
ocp4-stig-node-v2r3 [1]	Red Hat Openshift V2R3 向け国防情報システム局セキュリティー技術実装ガイド (DISA STIG)	ノード	DISA-STIG	**x86_64 ppc64le**	
rhcos4-stig[1]	国防情報システム局セキュリティー技術実装ガイド (DISA STIG)Red Hat Openshift [3]	ノード	DISA-STIG	**x86_64 ppc64le**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)
rhcos4-stig-v2r3	Red Hat Openshift V2R3 向け国防情報システム局セキュリティー技術実装ガイド (DISA STIG)	ノード	DISA-STIG	**x86_64 ppc64le**	Red Hat OpenShift Service on AWS with hosted control planes (ROSA with HCP)

1. **ocp4-stig**、**ocp4-stig-node**、**rhcos4-stig** プロファイルは、Compliance Operator で利用可能になると、DISA-STIG ベンチマークの最新バージョンを維持します。DISA-STIG V2R3 などの特定のバージョンに準拠する場合は、**ocp4-stig-v2r3** および **ocp4-stig-node-v2r3** プロファイルを使用します。

2. ノードプロファイルは、関連するプラットフォームプロファイルとともに使用する必要があります。詳細は、**Compliance Operator のプロファイルタイプ** を参照してください。

3. DISA-STIG V1R2 は DISA-STIG V2R3 に置き換えられました。最新のプロファイルを環境に適用することを推奨します。

5.6.1.1.9. 拡張コンプライアンスプロファイルについて

一部のコンプライアンスプロファイルには、業界のベストプラクティスに従う必要がある制御が含まれており、その結果、一部のプロファイルが他のプロファイルを拡張します。Center for Internet Security (CIS) のベストプラクティスと National Institute of Standards and Technology (NIST) のセキュリティーフレームワークを組み合わせることで、セキュアな準拠環境を実現するパスが確立されます。

たとえば NIST の High-Impact プロファイルおよび Moderate-Impact プロファイルは、コンプライアンスを達成するために CIS プロファイルを拡張します。その結果、拡張されたコンプライアンスプロファイルを使用することで、1 つのクラスターで両方のプロファイルを実行する必要がなくなります。

表5.9 プロファイルの拡張

プロファイル	拡張対象
ocp4-pci-dss	ocp4-cis
ocp4-pci-dss-node	ocp4-cis-node
ocp4-high	ocp4-cis
ocp4-high-node	ocp4-cis-node
ocp4-moderate	ocp4-cis
ocp4-moderate-node	ocp4-cis-node
ocp4-nerc-cip	ocp4-moderate
ocp4-nerc-cip-node	ocp4-moderate-node

5.6.1.1.10. Compliance Operator のプロファイルタイプ

Compliance Operator のルールはプロファイルとしてまとめられます。プロファイルは、OpenShift Container Platform のプラットフォームまたはノードを対象としています。一部のベンチマークには、**rhcos4** ノードプロファイルが含まれています。

プラットフォーム

プラットフォームプロファイルは、OpenShift Container Platform クラスターのコンポーネントを評価します。たとえば、プラットフォームレベルのルールで、APIServer 設定で強力な暗号化アルゴリズムが使用されているかどうかを確認できます。

ノード

ノードプロファイルは、各ホストの OpenShift または RHCOS 設定を評価します。**ocp4** ノードプロファイルと **rhcos4** ノードプロファイルの 2 つのノードプロファイルを使用できます。**ocp4** ノードプロファイルは、各ホストの OpenShift 設定を評価します。たとえば、**kubeconfig** ファイルにコンプライアンス標準に適合した正しい権限があるかどうかを確認できます。**rhcos4** ノードプロファイルは、各ホストの Red Hat Enterprise Linux CoreOS (RHCOS) 設定を評価します。たとえば、SSHD サービスがパスワードログインを無効にするように設定されているかどうかを確認できます。

重要

ノードプロファイルとプラットフォームプロファイル (PCI-DSS など) を含むベンチマークの場合、OpenShift Container Platform 環境で両方のプロファイルを実行する必要があります。

ocp4 プラットフォーム、**ocp4** ノード、および **rhcos4** ノードプロファイルを含むベンチマーク (FedRAMP High など) の場合、OpenShift Container Platform 環境で 3 つのプロファイルをすべて実行する必要があります。

注記

多数のノードを持つクラスターでは、**ocp4** ノードと **rhcos4** ノードの両方のスキャンが完了するまでに長い時間がかかる可能性があります。

5.6.2. Compliance Operator のスキャン

Compliance Operator を使用してコンプライアンススキャンを実行するには、**ScanSetting** および **ScanSettingBinding** API を使用することを推奨します。これらの API オブジェクトの詳細は、以下を実行します。

```
$ oc explain scansettings
```

または

```
$ oc explain scansettingbindings
```

5.6.2.1. コンプライアンススキャンの実行

Center for Internet Security (CIS) プロファイルを使用してスキャンを実行できます。便宜上、Compliance Operator は起動時に適切なデフォルト値を持つ **ScanSetting** オブジェクトを作成します。この **ScanSetting** オブジェクトの名前は **default** です。

注記

オールインワンのコントロールプレーンノードとワーカーノードの場合、コンプライアンススキャンはワーカーノードとコントロールプレーンノードで 2 回実行されます。コンプライアンススキャンは、一貫性のないスキャン結果を生成する可能性があります。**ScanSetting** オブジェクトで単一のロールのみを定義することにより、一貫性のない結果を回避できます。

重要

Compliance Operator のスキャンでは、コントロールプレーンが **aarch64** または **x86** CPU を使用しているかどうかに関係なく、マルチアーキテクチャーコンピュートマシンを含むクラスターで **INCONSISTENT** が報告されます。これは、同じルールが各アーキテクチャーで異なる動作をすることが原因です。これが発生するのは、Compliance Operator が複数のノードの結果を 1 つの結果に集約するノードスキャンの場合だけです。

一貫性のないスキャン結果の詳細は、Compliance Operator shows INCONSISTENT scan result with worker node を参照してください。

手順

1. 次のコマンドを実行して、**ScanSetting** オブジェクトを検査します。

```
$ oc describe scansettings default -n openshift-compliance
```

出力例

```
Name:             default
Namespace:        openshift-compliance
Labels:           <none>
Annotations:      <none>
API Version:      compliance.openshift.io/v1alpha1
Kind:             ScanSetting
```

```
Max Retry On Timeout:  3
Metadata:
  Creation Timestamp:  2024-07-16T14:56:42Z
  Generation:          2
  Resource Version:    91655682
  UID:                 50358cf1-57a8-4f69-ac50-5c7a5938e402
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce (1)
  Rotation:            3 (2)
  Size:                1Gi (3)
  Storage Class Name:  standard (4)
  Tolerations:
    Effect:            NoSchedule
    Key:               node-role.kubernetes.io/master
    Operator:          Exists
    Effect:            NoExecute
    Key:               node.kubernetes.io/not-ready
    Operator:          Exists
    Toleration Seconds:  300
    Effect:            NoExecute
    Key:               node.kubernetes.io/unreachable
    Operator:          Exists
    Toleration Seconds:  300
    Effect:            NoSchedule
    Key:               node.kubernetes.io/memory-pressure
    Operator:          Exists
Roles:
  master (5)
  worker (6)
Scan Tolerations: (7)
  Operator:            Exists
Schedule:              0 1 * * * (8)
Show Not Applicable:  false
Strict Node Scan:     true
Suspend:              false
Timeout:              30m
Events:               <none>
```

1. Compliance Operator は、スキャンの結果が含まれる永続ボリューム (PV) を作成します。デフォルトで、Compliance Operator はクラスターに設定されるストレージクラスについて何らかの仮定をすることができないため、PV はアクセスモード **ReadWriteOnce** を使用します。さらに、**ReadWriteOnce** アクセスモードはほとんどのクラスターで利用できます。スキャン結果を取得する必要がある場合は、ボリュームもバインドするヘルパー Pod を使用して実行できます。**ReadWriteOnce** アクセスモードを使用するボリュームは、一度に 1 つの Pod のみがマウントできるため、必ずヘルパー Pod を削除してください。そうでない場合は、Compliance Operator は後続のスキャンのためにボリュームを再利用できません。

2. Compliance Operator は、後続の 3 つのスキャンの結果をボリュームに保持し、古いスキャンはローテーションされます。

3. Compliance Operator はスキャンの結果用に 1 GB のストレージを割り当てます。

4 **scansetting.rawResultStorage.storageClassName** フィールドは、raw の結果を保存するための **PersistentVolumeClaim** オブジェクトを作成するときに使用する

5 6 スキャン設定がクラスターノードをスキャンするプロファイルを使用する場合は、これらのノードロールをスキャンします。

7 デフォルトのスキャン設定オブジェクトは、すべてのノードをスキャンします。

8 デフォルトのスキャン設定オブジェクトは、毎日 01:00 にスキャンを実行します。

デフォルトのスキャン設定の代わりに、以下の設定を含む **default-auto-apply** を使用できます。

```
Name:                  default-auto-apply
Namespace:             openshift-compliance
Labels:                <none>
Annotations:           <none>
API Version:           compliance.openshift.io/v1alpha1
Auto Apply Remediations:  true 1
Auto Update Remediations:  true 2
Kind:                  ScanSetting
Metadata:
  Creation Timestamp:  2022-10-18T20:21:00Z
  Generation:          1
  Managed Fields:
    API Version:  compliance.openshift.io/v1alpha1
    Fields Type:  FieldsV1
    fieldsV1:
      f:autoApplyRemediations:
      f:autoUpdateRemediations:
      f:rawResultStorage:
        .:
        f:nodeSelector:
          .:
          f:node-role.kubernetes.io/master:
        f:pvAccessModes:
        f:rotation:
        f:size:
        f:tolerations:
      f:roles:
      f:scanTolerations:
      f:schedule:
      f:showNotApplicable:
      f:strictNodeScan:
    Manager:         compliance-operator
    Operation:       Update
    Time:            2022-10-18T20:21:00Z
  Resource Version:  38840
  UID:               8cb0967d-05e0-4d7a-ac1c-08a7f7e89e84
Raw Result Storage:
  Node Selector:
    node-role.kubernetes.io/master:
  Pv Access Modes:
    ReadWriteOnce
  Rotation:  3
```

```
    Size:      1Gi
    Tolerations:
      Effect:            NoSchedule
      Key:               node-role.kubernetes.io/master
      Operator:            Exists
      Effect:            NoExecute
      Key:               node.kubernetes.io/not-ready
      Operator:            Exists
      Toleration Seconds:  300
      Effect:            NoExecute
      Key:               node.kubernetes.io/unreachable
      Operator:            Exists
      Toleration Seconds:  300
      Effect:            NoSchedule
      Key:               node.kubernetes.io/memory-pressure
      Operator:            Exists
  Roles:
    master
    worker
  Scan Tolerations:
    Operator:          Exists
  Schedule:            0 1 * * *
  Show Not Applicable:  false
  Strict Node Scan:     true
  Events:              <none>
```

1 2 **autoUpdateRemediations** および **autoApplyRemediations** フラグを **true** に設定すると、追加の手順なしに自動修復する **ScanSetting** オブジェクトを簡単に作成できます。

2. デフォルトの **ScanSetting** オブジェクトにバインドし、**cis** および **cis-node** プロファイルを使用してクラスターをスキャンする **ScanSettingBinding** オブジェクトを作成します。以下に例を示します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis-compliance
  namespace: openshift-compliance
profiles:
  - name: ocp4-cis-node
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
  - name: ocp4-cis
    kind: Profile
    apiGroup: compliance.openshift.io/v1alpha1
settingsRef:
  name: default
  kind: ScanSetting
  apiGroup: compliance.openshift.io/v1alpha1
```

3. 以下を実行して **ScanSettingBinding** オブジェクトを作成します。

```
$ oc create -f <file-name>.yaml -n openshift-compliance
```

プロセスのこの時点で、**ScanSettingBinding** オブジェクトは調整され、**Binding** および **Bound** 設定に基づいて調整されます。Compliance Operator は **ComplianceSuite** オブジェクトおよび関連付けられる **ComplianceScan** オブジェクトを作成します。

4. 以下を実行してコンプライアンススキャンの進捗を確認します。

```
$ oc get compliancescan -w -n openshift-compliance
```

スキャンはスキャンフェーズに進み、完了すると最終的に **DONE** フェーズに移行します。ほとんどの場合、スキャンの結果は **NON-COMPLIANT** になります。スキャン結果を確認し、クラスターがコンプライアンスに基づくように修復の適用を開始することができます。詳細は、Compliance Operator **修復の管理** を参照してください。

5.6.2.2. 結果に関するカスタムストレージサイズの設定

ComplianceCheckResult などのカスタムリソースは、スキャンされたすべてのノード間での単一チェックの集計された結果を表しますが、スキャナーによって生成される未加工の結果を確認することが役に立つ場合もあります。未加工の結果は ARF 形式で生成され、サイズが大きくなる可能性がありますが (ノードあたり数十メガバイト)、それらを **etcd** のキーと値のストアでサポートされる Kubernetes リソースに保存することは実際的ではありません。すべてのスキャンは、1GB のサイズにデフォルト設定される永続ボリューム (PV) を作成します。環境によっては、PV サイズを適宜増やす必要がある場合があります。これは、**ScanSetting** および **ComplianceScan** リソースの両方に公開される **rawResultStorage.size** 属性を使用して実行されます。

関連するパラメーターは **rawResultStorage.rotation** であり、これは古いスキャンのローテーションが行われる前に保持されるスキャンの数を制御します。デフォルト値は 3 です。ローテーションポリシーを 0 に設定するとローテーションは無効になります。デフォルトのローテーションポリシーと、未加工の ARF スキャンレポートにつき 100 MB を見積もり、お使いの環境に適した PV サイズを計算できます。

5.6.2.2.1. カスタム結果ストレージ値の使用

OpenShift Container Platform は各種のパブリッククラウドまたはベアメタルにデプロイできるので、Compliance Operator は利用可能なストレージ設定を判別できません。デフォルトで、Compliance Operator はクラスターのデフォルトのストレージクラスを使用して結果を保存するために PV の作成を試行しますが、カスタムストレージクラスは **rawResultStorage.StorageClassName** 属性を使用して設定できます。

重要

クラスターがデフォルトのストレージクラスを指定しない場合、この属性を設定する必要があります。

標準ストレージクラスを使用するように **ScanSetting** カスタムリソースを設定し、サイズが 10GB の永続ボリュームを作成し、最後の 10 件の結果を保持します。

ScanSetting CR の例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
```

```
    storageClassName: standard
    rotation: 10
    size: 10Gi
  roles:
  - worker
  - master
  scanTolerations:
  - effect: NoSchedule
    key: node-role.kubernetes.io/master
    operator: Exists
  schedule: '0 1 * * *'
```

5.6.2.3. ワーカーノードでの結果サーバー Pod のスケジューリング

結果サーバー Pod は、生の Asset Reporting Format (ARF) スキャン結果を格納する永続ボリューム (PV) をマウントします。**nodeSelector** 属性および **tolerations** 属性を使用すると、結果サーバー Pod の場所を設定できます。

これは、コントロールプレーンノードが永続ボリュームをマウントすることを許可されていない環境で役立ちます。

手順

- Compliance Operator 用の **ScanSetting** カスタムリソース (CR) を作成します。

  a. **ScanSetting** CR を定義し、YAML ファイルを保存します (例: **rs-workers.yaml**)。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: rs-on-workers
  namespace: openshift-compliance
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: "" 1
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists 2
roles:
- worker
- master
scanTolerations:
  - operator: Exists
schedule: 0 1 * * *
```

1 Compliance Operator は、このノードを使用してスキャン結果を ARF 形式で保存します。

2 結果のサーバー Pod は、すべてのテイントを許容します。

  b. **ScanSetting** CR を作成するには、次のコマンドを実行します。

```
$ oc create -f rs-workers.yaml
```

検証

- **ScanSetting** オブジェクトが作成されたことを確認するには、次のコマンドを実行します。

```
$ oc get scansettings rs-on-workers -n openshift-compliance -o yaml
```

出力例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  creationTimestamp: "2021-11-19T19:36:36Z"
  generation: 1
  name: rs-on-workers
  namespace: openshift-compliance
  resourceVersion: "48305"
  uid: 43fdfc5f-15a7-445a-8bbc-0e4a160cd46e
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/worker: ""
  pvAccessModes:
  - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
  - operator: Exists
roles:
- worker
- master
scanTolerations:
- operator: Exists
schedule: 0 1 * * *
strictNodeScan: true
```

5.6.2.4. ScanSetting カスタムリソース

ScanSetting カスタムリソースでは、scan limits 属性を使用して、スキャナー Pod のデフォルトの CPU およびメモリー制限をオーバーライドできるようになりました。Compliance Operator は、スキャナーコンテナーに 500Mi メモリー、100m CPU のデフォルトを使用し、**api-resource-collector** コンテナーに 100m CPU の 200Mi メモリーを使用します。Operator のメモリー制限を設定するには、OLM または Operator デプロイメント自体を介してインストールされている場合は **Subscription** オブジェクトを変更します。

Compliance Operator のデフォルトの CPU およびメモリーの制限を増やすには、**Compliance Operator のリソース制限の引き上げ** を参照してください。

重要

デフォルトの制限が十分ではなく、Operator またはスキャナー Pod が Out Of Memory (OOM) プロセスによって終了した場合は、Compliance Operator またはスキャナー Pod のメモリー制限を増やす必要があります。詳細は、Compliance Operator のリソース制限の引き上げ を参照してください。

5.6.2.5. Hosted Control Plane の管理クラスターの設定

独自のホステッドコントロールプレーンまたは Hypershift 環境をホストしており、管理クラスターからホステッドクラスターをスキャンする場合は、ターゲットであるホステッドクラスターの名前とプレフィックス namespace を設定する必要があります。これは、**TailoredProfile** を作成することで行なえます。

重要

この手順は、独自の Hosted Control Plane 環境を管理するユーザーにのみ適用されます。

注記

Hosted Control Plane の管理クラスターでは、**ocp4-cis** および **ocp4-pci-dss** プロファイルのみサポートされます。

前提条件

- Compliance Operator が管理クラスターにインストールされている。

手順

1. 次のコマンドを実行して、スキャン対象のホステッドクラスターの **name** と **namespace** を取得します。

```
$ oc get hostedcluster -A
```

出力例

```
NAMESPACE      NAME                VERSION   KUBECONFIG
PROGRESS   AVAILABLE   PROGRESSING   MESSAGE
local-cluster   79136a1bdb84b3c13217   4.13.5    79136a1bdb84b3c13217-admin-kubeconfig
  Completed   True        False         The hosted control plane is available
```

2. 管理クラスターで、スキャンプロファイルを拡張する **TailoredProfile** を作成し、スキャンするホステッドクラスターの名前と namespace を定義します。

例: management-tailoredprofile.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: hypershift-cisk57aw88gry
  namespace: openshift-compliance
spec:
```

```
description: This profile test required rules
extends: ocp4-cis (1)
title: Management namespace profile
setValues:
- name: ocp4-hypershift-cluster
  rationale: This value is used for HyperShift version detection
  value: 79136a1bdb84b3c13217 (2)
- name: ocp4-hypershift-namespace-prefix
  rationale: This value is used for HyperShift control plane namespace detection
  value: local-cluster (3)
```

1. 変数。Hosted Control Plane の管理クラスターでは、**ocp4-cis** および **ocp4-pci-dss** プロファイルのみサポートされます。
2. **value** は、前の手順の出力の **NAME** です。
3. **value** は、直前のステップの出力の **NAMESPACE** です。

3. **TailoredProfile** を作成します。

```
$ oc create -n openshift-compliance -f mgmt-tp.yaml
```

5.6.2.6. リソース要求および制限の適用

kubelet が Pod の一部としてコンテナーを起動すると、kubelet はそのコンテナーの要求および制限をメモリーおよび CPU の要求および制限をコンテナーランタイムに渡します。Linux では、コンテナーランタイムは、定義した制限を適用して有効にするカーネル cgroup を設定します。

CPU 制限は、コンテナーが使用できる CPU 時間を定義します。各スケジューリング期間中、Linux カーネルはこの制限を超えるかどうかを確認します。その場合、カーネルは、cgroup の実行を再開できるようにするまで待機します。

複数の異なるコンテナー (cgroup) を競合するシステムで実行する場合、CPU 要求が大きいワークロードには、要求が小さいワークロードよりも多くの CPU 時間が割り当てられます。メモリー要求は Pod のスケジューリング時に使用されます。cgroups v2 を使用するノードでは、コンテナーランタイムがメモリーリクエストをヒントとして使用して、**memory.min** および **memory.low** の値を設定する場合があります。

コンテナーがこの制限を超えるメモリーを割り当てようとすると、Linux カーネルのメモリー不足サブシステムがアクティブになり、メモリーを割り当てようとしたコンテナー内のプロセスの 1 つを停止して介入します。Pod またはコンテナーのメモリー制限は、emptyDir などのメモリーベースのボリュームのページにも適用できます。

kubelet は、ローカルの一時ストレージとしてではなく、コンテナーメモリーが使用されているときに **tmpfs emptyDir** ボリュームを追跡します。コンテナーがメモリー要求を超え、それが実行されているノードが全体的にメモリー不足になった場合、Pod のコンテナーが削除される可能性があります。

重要

コンテナーは、長期間にわたって CPU 制限を超えることはできません。コンテナーのランタイムは、CPU 使用率が過剰に使用されている場合も Pod またはコンテナーを停止しません。リソース制限のためにコンテナーをスケジュールできないか、強制終了されているかを判断するには、**Compliance Operator のトラブルシューティング** を参照してください。

5.6.2.7. コンテナーリソース要求を使用した Pod のスケジューリング

Pod が作成されると、スケジューラーは Pod を実行するノードを選択します。各ノードには、リソースタイプごとに、Pod に提供できる CPU とメモリーの最大容量があります。スケジューラーは、スケジュールされたコンテナーのリソース要求の合計が、各リソースタイプの容量ノードよりも少なくなるようにします。

ノードのメモリーまたは CPU リソースの使用率が非常に低い場合でも、容量チェックでノードのリソース不足を防ぐことができない場合、スケジューラーはノードへの Pod の配置を拒否することがあります。

コンテナーごとに、以下のリソース制限および要求を指定できます。

```
spec.containers[].resources.limits.cpu
spec.containers[].resources.limits.memory
spec.containers[].resources.limits.hugepages-<size>
spec.containers[].resources.requests.cpu
spec.containers[].resources.requests.memory
spec.containers[].resources.requests.hugepages-<size>
```

個々のコンテナーに対してのみ要求と制限を指定できますが、Pod の全体的なリソース要求と制限を考慮することも役立ちます。特定のリソースの場合には、コンテナーリソースの要求または制限は、Pod 内にあるコンテナーごとに割り当てられた、対象タイプのリソース要求または制限を合計したものです。

コンテナーリソース要求および制限の例

```
apiVersion: v1
kind: Pod
metadata:
  name: frontend
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
  - name: app
    image: images.my-company.example/app:v4
    resources:
      requests: 1
        memory: "64Mi"
        cpu: "250m"
      limits: 2
        memory: "128Mi"
        cpu: "500m"
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
  - name: log-aggregator
    image: images.my-company.example/log-aggregator:v6
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
```

```
    limits:
      memory: "128Mi"
      cpu: "500m"
  securityContext:
    allowPrivilegeEscalation: false
    capabilities:
      drop: [ALL]
```

1 コンテナーは 64 Mi のメモリーと 250 m CPU を要求しています。

2 コンテナーの制限は 128 Mi のメモリーと 500 m CPU です。

5.6.3. Compliance Operator の調整

Compliance Operator には、追加の設定なしで使用できるプロファイルが同梱されますが、それらは組織のニーズおよび要件を満たすために変更される必要があります。プロファイルを変更するプロセスは、**テーラリング** と呼ばれます。

Compliance Operator は、**TailoredProfile** オブジェクトを提供し、プロファイルを調整します。

5.6.3.1. 調整されたプロファイルの新規作成

TailoredProfile オブジェクトを使用して、調整されたプロファイルをゼロから作成できます。適切な **title** と **description** を設定し、**extends** フィールドを空のままにします。このカスタムプロファイルが生成するスキャンのタイプを Compliance Operator に指定します。

- ノードのスキャン: オペレーティングシステムをスキャンします。
- プラットフォームスキャン: OpenShift Container Platform の設定をスキャンします。

手順

- **TailoredProfile** オブジェクトに以下のアノテーションを設定します。

例: new-profile.yaml

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: new-profile
  annotations:
    compliance.openshift.io/product-type: Node 1
spec:
  extends: ocp4-cis-node 2
  description: My custom profile 3
  title: Custom profile 4
  enableRules:
    - name: ocp4-etcd-unique-ca
      rationale: We really need to enable this
  disableRules:
    - name: ocp4-file-groupowner-cni-conf
      rationale: This does not apply to the cluster
```

1. それに応じて **Node** または **Platform** を設定します。
2. **extends** フィールドはオプションです。
3. **description** フィールドを使用して、新しい **TailoredProfile** オブジェクトの機能を記述します。
4. **title** フィールドで、**TailoredProfile** オブジェクトにタイトルを付けます。

> **注記**
>
> **TailoredProfile** オブジェクトの **name** フィールドに **-node** 接尾辞を追加すると、**Node** 製品タイプのアノテーションを追加するのと同様の効果があり、オペレーティングシステムのスキャンが生成されます。

5.6.3.2. 調整されたプロファイルを使用した既存の ProfileBundles の拡張

TailoredProfile CR は最も一般的なテーラリング操作を有効にする一方で、XCCDF 標準は OpenSCAP プロファイルの調整におけるより多くの柔軟性を提供します。さらに、組織が以前に OpenScap を使用していた場合、既存の XCCDF テーラリングファイルが存在し、これを再利用できる可能性があります。

ComplianceSuite オブジェクトには、カスタムのテーラリングファイルにポイントできるオプションの **TailoringConfigMap** 属性が含まれます。**TailoringConfigMap** 属性の値は設定マップの名前です。これには、**tailoring.xml** というキーが含まれる必要があり、このキーの値はテーラリングのコンテンツです。

手順

1. Red Hat Enterprise Linux CoreOS (RHCOS) **ProfileBundle** の利用可能なルールを参照します。

```
$ oc get rules.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

2. 同じ **ProfileBundle** で利用可能な変数を参照します。

```
$ oc get variables.compliance -n openshift-compliance -l compliance.openshift.io/profile-bundle=rhcos4
```

3. **nist-moderate-modified** という名前の調整されたプロファイルを作成します。
   a. 調整されたプロファイル **nist-moderate-modified** に追加するルールを選択します。この例では、2 つのルールを無効にし、1 つの値を変更することにより、**rhcos4-moderate** プロファイルを拡張します。**rationale** 値を使用して、これらの変更が加えられた理由を記述します。

   new-profile-node.yaml の例

```yaml
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: nist-moderate-modified
spec:
```

```
extends: rhcos4-moderate
description: NIST moderate profile
title: My modified NIST moderate profile
disableRules:
- name: rhcos4-file-permissions-var-log-messages
  rationale: The file contains logs of error messages in the system
- name: rhcos4-account-disable-post-pw-expiration
  rationale: No need to check this as it comes from the IdP
setValues:
- name: rhcos4-var-selinux-state
  rationale: Organizational requirements
  value: permissive
```

表5.10 スペック変数の属性

属性	説明
extends	この **TailoredProfile** がビルドされる **Profile** オブジェクトの名前。
title	人間が判読できる **TailoredProfile** のタイトル。
disableRules	名前および理論的根拠のペアのリスト。名前ごとに、無効にするルールオブジェクトの名前を参照します。理論的根拠の値は、人間が判読できるテキストで、ルールが無効になっている理由を記述します。
manualRules	名前および理論的根拠のペアのリスト。手動ルールを追加すると、チェック結果のステータスは常に **manual** になり、修復は生成されません。この属性は自動であり、手動ルールとして設定されている場合、デフォルトでは値がありません。
enableRules	名前および理論的根拠のペアのリスト。名前ごとに、有効にするルールオブジェクトの名前を参照します。理論的根拠の値は、人間が判読できるテキストで、ルールが有効になっている理由を記述します。
description	**TailoredProfile** を記述する人間が判読できるテキスト。
setValues	名前、理論的根拠、および値のグループ化のリスト。各名前は値セットの名前を参照します。この理論的根拠は、値セットを記述する、人間が判読できるテキストです。この値は実際の設定です。

b. **tailoredProfile.spec.manualRules** 属性を追加します。

tailoredProfile.spec.manualRules.yaml 例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: TailoredProfile
metadata:
  name: ocp4-manual-scc-check
```

```
spec:
  extends: ocp4-cis
  description: This profile extends ocp4-cis by forcing the SCC check to always return
MANUAL
  title: OCP4 CIS profile with manual SCC check
  manualRules:
    - name: ocp4-scc-limit-container-allowed-capabilities
      rationale: We use third party software that installs its own SCC with extra privileges
```

c. **TailoredProfile** オブジェクトを作成します。

```
$ oc create -n openshift-compliance -f new-profile-node.yaml (1)
```

(1) **TailoredProfile** オブジェクトは、デフォルトの **openshift-compliance** namespace で作成されます。

出力例

```
tailoredprofile.compliance.openshift.io/nist-moderate-modified created
```

4. **ScanSettingBinding** オブジェクトを定義して、新しい調整されたプロファイル **nist-moderate-modified** をデフォルトの **ScanSetting** オブジェクトにバインドします。

new-scansettingbinding.yaml の例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: nist-moderate-modified
profiles:
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: Profile
    name: ocp4-moderate
  - apiGroup: compliance.openshift.io/v1alpha1
    kind: TailoredProfile
    name: nist-moderate-modified
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

5. **ScanSettingBinding** オブジェクトを作成します。

```
$ oc create -n openshift-compliance -f new-scansettingbinding.yaml
```

出力例

```
scansettingbinding.compliance.openshift.io/nist-moderate-modified created
```

5.6.4. Compliance Operator の未加工の結果の取得

OpenShift Container Platform クラスターのコンプライアンスを証明する際に、監査目的でスキャンの結果を提供する必要がある場合があります。

5.6.4.1. 永続ボリュームからの Compliance Operator の未加工の結果の取得

手順

Compliance Operator は、未加工の結果を生成し、これを永続ボリュームに保存します。これらの結果は Asset Reporting Format (ARF) で生成されます。

1. **ComplianceSuite** オブジェクトを確認します。

```
$ oc get compliancesuites nist-moderate-modified \
-o json -n openshift-compliance | jq '.status.scanStatuses[].resultsStorage'
```

出力例

```
{
    "name": "ocp4-moderate",
    "namespace": "openshift-compliance"
}
{
    "name": "nist-moderate-modified-master",
    "namespace": "openshift-compliance"
}
{
    "name": "nist-moderate-modified-worker",
    "namespace": "openshift-compliance"
}
```

これは、未加工の結果にアクセスできる永続ボリューム要求を表示します。

2. 結果のいずれかの名前と namespace を使用して、未加工データの場所を確認します。

```
$ oc get pvc -n openshift-compliance rhcos4-moderate-worker
```

出力例

```
NAME                   STATUS   VOLUME                                     CAPACITY   ACCESS MODES
STORAGECLASS   AGE
rhcos4-moderate-worker   Bound pvc-548f6cfe-164b-42fe-ba13-a07cfbc77f3a   1Gi     RWO
gp2         92m
```

3. ボリュームをマウントする Pod を起動し、結果をコピーして、未加工の結果をフェッチします。

```
$ oc create -n openshift-compliance -f pod.yaml
```

pod.yaml の例

```
apiVersion: "v1"
kind: Pod
metadata:
```

```
  name: pv-extract
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: pv-extract-pod
      image: registry.access.redhat.com/ubi9/ubi
      command: ["sleep", "3000"]
      volumeMounts:
      - mountPath: "/workers-scan-results"
        name: workers-scan-vol
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
  volumes:
    - name: workers-scan-vol
      persistentVolumeClaim:
        claimName: rhcos4-moderate-worker
```

4. Pod の実行後に、結果をダウンロードします。

```
$ oc cp pv-extract:/workers-scan-results -n openshift-compliance .
```

重要

永続ボリュームをマウントする Pod を起動すると、要求は **Bound** として保持されます。使用中のボリュームのストレージクラスのパーミッションが **ReadWriteOnce** に設定されている場合、ボリュームは一度に 1 つの Pod によってのみマウント可能です。完了したら Pod を削除する必要があります。そうしないと、Operator は Pod をスケジュールし、継続して結果をこの場所に保存し続けることができなくなります。

5. 展開が完了した後に、Pod を削除できます。

```
$ oc delete pod pv-extract -n openshift-compliance
```

5.6.5. Compliance Operator の結果と修復の管理

それぞれの **ComplianceCheckResult** は、1 つのコンプライアンスルールチェックの結果を表します。ルールを自動的に修復できる場合、**ComplianceCheckResult** によって所有される、同じ名前を持つ **ComplianceRemediation** オブジェクトが作成されます。修復が要求されない限り、修復は自動的に適用されません。これにより、OpenShift Container Platform の管理者は修復内容を確認し、検証後にのみ修復を適用することができます。

重要

Federal Information Processing Standards (FIPS) コンプライアンスを完全に修復するには、クラスターの FIPS モードを有効にする必要があります。FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL での FIPS モードの設定の詳細は、FIPS モードでのシステムのインストール を参照してください。

FIPS モードは、次のアーキテクチャーでサポートされています。

- **x86_64**
- **ppc64le**
- **s390x**

5.6.5.1. コンプライアンスチェック結果のフィルター

デフォルトで、**ComplianceCheckResult** オブジェクトには、チェックのクエリーおよび結果の生成後に次のステップを決定することを可能にする便利なラベルが複数付けられます。

特定のスイートに属するチェックをリスト表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
  -l compliance.openshift.io/suite=workers-compliancesuite
```

特定のスキャンに属するチェックをリスト表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l compliance.openshift.io/scan=workers-scan
```

すべての **ComplianceCheckResult** オブジェクトが **ComplianceRemediation** オブジェクトを作成する訳ではありません。自動的に修復できる **ComplianceCheckResult** オブジェクトのみになります。**ComplianceCheckResult** オブジェクトには、**compliance.openshift.io/automated-remediation** ラベルでラベル付けされる場合に関連する修復が含まれます。修復の名前はチェックの名前と同じです。

自動的に修復できる障害のあるチェックをすべてリスト表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/automated-remediation'
```

失敗したすべてのチェックを重大度順に一覧表示します。

```
$ oc get compliancecheckresults -n openshift-compliance \
-l 'compliance.openshift.io/check-status=FAIL,compliance.openshift.io/check-severity=high'
```

出力例

```
NAME                                                  STATUS   SEVERITY
nist-moderate-modified-master-configure-crypto-policy        FAIL     high
nist-moderate-modified-master-coreos-pti-kernel-argument       FAIL     high
nist-moderate-modified-master-disable-ctrlaltdel-burstaction   FAIL     high
nist-moderate-modified-master-disable-ctrlaltdel-reboot        FAIL     high
```

```
nist-moderate-modified-master-enable-fips-mode                FAIL     high
nist-moderate-modified-master-no-empty-passwords              FAIL     high
nist-moderate-modified-master-selinux-state                   FAIL     high
nist-moderate-modified-worker-configure-crypto-policy         FAIL     high
nist-moderate-modified-worker-coreos-pti-kernel-argument      FAIL     high
nist-moderate-modified-worker-disable-ctrlaltdel-burstaction  FAIL     high
nist-moderate-modified-worker-disable-ctrlaltdel-reboot       FAIL     high
nist-moderate-modified-worker-enable-fips-mode                FAIL     high
nist-moderate-modified-worker-no-empty-passwords              FAIL     high
nist-moderate-modified-worker-selinux-state                   FAIL     high
ocp4-moderate-configure-network-policies-namespaces           FAIL     high
ocp4-moderate-fips-mode-enabled-on-all-nodes                  FAIL     high
```

手動で修復する必要のある障害のあるチェックをすべてリスト表示します。

```
$ oc get -n openshift-compliance compliancecheckresults \
-l 'compliance.openshift.io/check-status=FAIL,!compliance.openshift.io/automated-remediation'
```

手動による修復の手順は、通常 **ComplianceCheckResult** オブジェクトの **description** 属性に保存されます。

表5.11 ComplianceCheckResult Status

ComplianceCheckResult Status	説明
PASS	コンプライアンスチェックが完了し、パスしました。
FAIL	コンプライアンスチェックが完了するまで実行され、失敗しました。
INFO	コンプライアンスチェックが完了するまで実行され、エラーと見なされるほど深刻ではないものが見つかりました。
MANUAL	コンプライアンスチェックには、成功または失敗を自動的に評価する方法がないため、手動でチェックする必要があります。
INCONSISTENT	コンプライアンスチェックは、さまざまなソース (通常はクラスターノード) からのさまざまな結果を報告します。
ERROR	コンプライアンスチェックは実行されましたが、正しく完了できませんでした。
NOT-APPLICABLE	該当しない、または選択されていないため、コンプライアンスチェックは実行されませんでした。

5.6.5.2. 修復の確認

ComplianceRemediation オブジェクト、および修復を所有する **ComplianceCheckResult** オブジェク

トの両方を確認します。**ComplianceCheckResult** オブジェクトには、チェック内容やサーバーの強化措置などの人間が判読できる記述、および重大度や関連するセキュリティーコントロールなどの他の **metadata** が含まれます。**ComplianceRemediation** オブジェクトは、**ComplianceCheckResult** に説明されている問題を修正する方法を表します。最初のスキャン後、**MissingDependencies** 状態の修復を確認します。

以下は、**sysctl-net-ipv4-conf-all-accept-redirects** というチェックおよび修復の例です。この例では、**spec** および **status** のみを表示し、**metadata** は省略するように編集されています。

```
spec:
  apply: false
  current:
  object:
    apiVersion: machineconfiguration.openshift.io/v1
    kind: MachineConfig
    spec:
      config:
        ignition:
          version: 3.2.0
        storage:
          files:
            - path: /etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf
              mode: 0644
              contents:
                source: data:,net.ipv4.conf.all.accept_redirects%3D0
  outdated: {}
status:
  applicationState: NotApplied
```

修復ペイロードは **spec.current** 属性に保存されます。ペイロードには Kubernetes オブジェクトを使用することができますが、この修復はノードスキャンによって生成されたため、上記の例の修復ペイロードは **MachineConfig** オブジェクトになります。Platform スキャンでは、修復ペイロードは多くの場合 (**ConfigMap** や **Secret** オブジェクトなど) 異なる種類のオブジェクトになりますが、通常は、修復を適用については管理者が判断します。そうでない場合には、Compliance Operator には汎用の Kubernetes オブジェクトを操作するために非常に幅広いパーミッションのセットが必要になるためです。Platform チェックの修復例は、後ほど提供されます。

修復が適用される際に修復内容を正確に確認できるようにするために、**MachineConfig** オブジェクトの内容は設定の Ignition オブジェクトを使用します。形式に関する詳細は、Ignition 仕様 を参照してください。この例では、**spec.config.storage.files[0].path** 属性は、この修復によって作成されるファイル (**/etc/sysctl.d/75-sysctl_net_ipv4_conf_all_accept_redirects.conf**) を指定し、**spec.config.storage.files[0].contents.source** 属性はそのファイルの内容を指定します。

注記

ファイルの内容は URL でエンコードされます。

以下の Python スクリプトを使用して、コンテンツを表示します。

```
$ echo "net.ipv4.conf.all.accept_redirects%3D0" | python3 -c "import sys, urllib.parse;
print(urllib.parse.unquote(''.join(sys.stdin.readlines())))"
```

出力例

```
net.ipv4.conf.all.accept_redirects=0
```

重要

Compliance Operator は、修復の間に発生する可能性のある依存関係の問題を自動的に解決しません。正確な結果を確実に得るためには、修復が適用された後に再度スキャンする必要があります。

5.6.5.3. カスタマイズされたマシン設定プールを使用するときに修復を適用する

カスタム **MachineConfigPool** を作成するときは、**MachineConfigPool** にラベルを追加して、**KubeletConfig** に存在する **machineConfigPoolSelector** がそのラベルを **MachineConfigPool** と一致させることができるようにします。

重要

KubeletConfig ファイルでは、**protectKernelDefaults: false** を設定しないでください。Compliance Operator が修復の適用を完了した後、**MachineConfigPool** オブジェクトが予期せず一時停止解除に失敗する可能性があるためです。

手順

1. ノードを一覧表示します。

```
$ oc get nodes -n openshift-compliance
```

出力例

```
NAME                                       STATUS  ROLES  AGE    VERSION
ip-10-0-128-92.us-east-2.compute.internal  Ready   master 5h21m  v1.29.4
ip-10-0-158-32.us-east-2.compute.internal  Ready   worker 5h17m  v1.29.4
ip-10-0-166-81.us-east-2.compute.internal  Ready   worker 5h17m  v1.29.4
ip-10-0-171-170.us-east-2.compute.internal Ready   master 5h21m  v1.29.4
ip-10-0-197-35.us-east-2.compute.internal  Ready   master 5h22m  v1.29.4
```

2. ノードにラベルを追加します。

```
$ oc -n openshift-compliance \
label node ip-10-0-166-81.us-east-2.compute.internal \
node-role.kubernetes.io/<machine_config_pool_name>=
```

出力例

```
node/ip-10-0-166-81.us-east-2.compute.internal labeled
```

3. カスタム **MachineConfigPool** CR を作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfigPool
metadata:
  name: <machine_config_pool_name>
  labels:
```

```
    pools.operator.machineconfiguration.openshift.io/<machine_config_pool_name>: '' 1
spec:
  machineConfigSelector:
  matchExpressions:
  - {key: machineconfiguration.openshift.io/role, operator: In, values: [worker,
<machine_config_pool_name>]}
  nodeSelector:
  matchLabels:
    node-role.kubernetes.io/<machine_config_pool_name>: ""
```

1 **labels** フィールドは、マシン設定プール (MCP) に追加するラベル名を定義します。

4. MCP が正常に作成されたことを確認します。

```
$ oc get mcp -w
```

5.6.5.4. デフォルトの設定値をもとにした KubeletConfig ルールの評価

OpenShift Container Platform インフラストラクチャーには、実行時に不完全な設定ファイルが含まれる場合があり、ノードは、設定オプションが欠落している場合には、設定値がデフォルトであることを想定します。一部の設定オプションは、コマンドライン引数として渡すことができます。その結果、Compliance Operator はノード上の設定ファイルが完全かどうかを確認できません。これは、ルールチェックで使用されるオプションが欠落している可能性があるためです。

デフォルトの設定値がチェックに合格するといったフォールスネガティブの結果に陥らないように、Compliance Operator はノード/プロキシー API を使用してノードのプール内にあるノードごとに設定をフェッチし、次にノードプール内のノード全体で整合性が取れた設定オプションすべてをファイルに保存することで、このファイルが対象ノードプール内の全ノードの設定を表します。これにより、スキャン結果の精度が向上します。

デフォルトの **master** および **worker** ノードプール設定でこの機能を使用するために、追加の設定変更は必要ありません。

5.6.5.5. カスタムノードプールのスキャン

Compliance Operator は、各ノードプール設定のコピーを維持しません。Compliance Operator は、単一ノードプール内のすべてのノードについて一貫性のある設定オプションを設定ファイルの 1 つのコピーに集約します。次に、Compliance Operator は、特定のノードプールの設定ファイルを使用して、そのプール内のノードに対してルールを評価します。

手順

1. **ScanSettingBinding** CR に保存される **ScanSetting** オブジェクトに **example** ロールを追加します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
```

```
- worker
- master
- example
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

2. **ScanSettingBinding** CR を使用するスキャンを作成します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: cis
  namespace: openshift-compliance
profiles:
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis
- apiGroup: compliance.openshift.io/v1alpha1
  kind: Profile
  name: ocp4-cis-node
settingsRef:
  apiGroup: compliance.openshift.io/v1alpha1
  kind: ScanSetting
  name: default
```

検証

- Platform KubeletConfig ルールは、**Node**/**Proxy** オブジェクトを介してチェックされます。これらのルールは、以下のコマンドを実行して検索できます。

```
$ oc get rules -o json | jq '.items[] | select(.checkType == "Platform") | select(.metadata.name
| contains("ocp4-kubelet-")) | .metadata.name'
```

5.6.5.6. **KubeletConfig** サブプールの修復

KubeletConfig 修復ラベルは **MachineConfigPool** サブプールに適用できます。

手順

- ラベルをサブプール **MachineConfigPool** CR に追加します。

```
$ oc label mcp <sub-pool-name> pools.operator.machineconfiguration.openshift.io/<sub-
pool-name>=
```

5.6.5.7. 修復の適用

ブール値の属性 **spec.apply** は、Compliance Operator で修復を適用するかどうかを制御します。属性を **true** に設定すると、修復を適用することができます。

```
$ oc -n openshift-compliance \
patch complianceremediations/<scan-name>-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":true}}' --type=merge
```

Compliance Operator が適用された修復を処理した後に、**status.ApplicationState** 属性は **Applied** に、または正しくない場合には **Error** に切り替わります。マシン設定の修復が適用されると、その修復と他のすべての適用済みの修復が **75-$scan-name-$suite-name** という名前の **MachineConfig** オブジェクトにレンダリングされます。**MachineConfig** オブジェクトはその後 Machine Config Operator によってレンダリングされ、最終的に各ノードで実行されるマシン制御デーモンのインスタンスによってマシン設定プールのすべてのノードに適用されます。

Machine Config Operator が新規 **MachineConfig** オブジェクトをプール内のノードに適用すると、そのプールに属するすべてのノードが再起動されることに注意してください。これは、それぞれが複合の **75-$scan-name-$suite-name MachineConfig** オブジェクトを再度レンダリングする、複数の修復を適用する際に不都合が生じる可能性があります。修復をすぐに適用しないようにするには、**MachineConfigPool** オブジェクトの **.spec.paused** 属性を **true** に設定してマシン設定プールを一時停止できます。

Compliance Operator は修復を自動的に適用できます。**ScanSetting** の最上位のオブジェクトに **autoApplyRemediations: true** を設定します。

警告

修復の自動敵に適用するかどうかについては、慎重に考慮する必要があります。

重要

Compliance Operator は、修復の間に発生する可能性のある依存関係の問題を自動的に解決しません。正確な結果を確実に得るためには、修復が適用された後に再度スキャンする必要があります。

5.6.5.8. プラットフォームチェックの手動による修復

通常、プラットフォームスキャンをチェックする場合、以下の 2 つの理由のために管理者が手動で修復する必要があります。

- 設定する必要のある値は、常に自動的に判別できる訳ではありません。チェックのいずれかで許可されたレジストリーのリストが指定されることが必要になりますが、スキャナー側が組織が許可する必要のあるレジストリーを認識する方法はありません。
- 異なるチェックで異なる API オブジェクトが変更されるため、クラスター内のオブジェクトを変更するために自動化された修復で **root** またはスーパーユーザーアクセスを所有することが要求されますが、これは推奨されていません。

手順

1. 以下の例では、**ocp4-ocp-allowed-registries-for-import** ルールを使用します。これはデフォルトの OpenShift Container Platform のインストール時に失敗します。ルール **oc get rule.compliance/ocp4-ocp-allowed-registries-for-import -oyaml** を検査します。このルール

は、**allowedRegistriesForImport** 属性を設定して、ユーザーのイメージのインポートを許可するレジストリーを制限します。さらに、ルールの **warning** 属性はチェックされた API オブジェクトを表示するため、これを変更し、問題を修復できます。

```
$ oc edit image.config.openshift.io/cluster
```

出力例

```
apiVersion: config.openshift.io/v1
kind: Image
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2020-09-10T10:12:54Z"
  generation: 2
  name: cluster
  resourceVersion: "363096"
  selfLink: /apis/config.openshift.io/v1/images/cluster
  uid: 2dcb614e-2f8a-4a23-ba9a-8e33cd0ff77e
spec:
  allowedRegistriesForImport:
  - domainName: registry.redhat.io
status:
  externalRegistryHostnames:
  - default-route-openshift-image-registry.apps.user-cluster-09-10-12-
07.devcluster.openshift.com
  internalRegistryHostname: image-registry.openshift-image-registry.svc:5000
```

2. スキャンを再実行します。

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.6.5.9. 修復の更新

新しいバージョンのコンプライアンスコンテンツが使用されると、以前のバージョンとは異なる新しいバージョンの修復が提供される可能性があります。Compliance Operator は、適用される修復の以前のバージョンを保持します。OpenShift Container Platform の管理者は、確認し、適用する新規バージョンも通知されます。以前に適用されたものの、更新された ComplianceRemediation オブジェクトはそのステータスを **Outdated** に変更します。古いオブジェクトには簡単に検索できるようにラベルが付けられます。

以前に適用された修復内容は **ComplianceRemediation** オブジェクトの **spec.outdated** 属性に保存され、新規に更新された内容は **spec.current** 属性に保存されます。コンテンツを新しいバージョンに更新した後に、管理者は修復を確認する必要があります。**spec.outdated** 属性が存在する限り、これは結果として作成される **MachineConfig** オブジェクトをレンダリングするために使用されます。**spec.outdated** 属性が削除された後に、Compliance Operator は結果として生成される **MachineConfig** オブジェクトを再度レンダリングし、これにより Operator は設定をノードにプッシュします。

手順

1. 古い修復を検索します。

```
$ oc -n openshift-compliance get complianceremediations \
-l complianceoperator.openshift.io/outdated-remediation=
```

出力例

```
NAME                              STATE
workers-scan-no-empty-passwords   Outdated
```

現在適用されている修復は **Outdated** 属性に保存され、新規の、適用されていない修復は **Current** 属性に保存されます。新規バージョンに問題がなれば、**Outdated** フィールドを削除します。更新された内容を保持する必要がある場合には、**Current** および **Outdated** 属性を削除します。

2. 修復の新しいバージョンを適用します。

```
$ oc -n openshift-compliance patch complianceremediations workers-scan-no-empty-passwords \
--type json -p '[{"op":"remove", "path":/spec/outdated}]'
```

3. 修復の状態は、**Outdated** から **Applied** に切り替わります。

```
$ oc get -n openshift-compliance complianceremediations workers-scan-no-empty-passwords
```

出力例

```
NAME                              STATE
workers-scan-no-empty-passwords   Applied
```

4. ノードは新規の修復バージョンを適用し、再起動します。

重要

Compliance Operator は、修復の間に発生する可能性のある依存関係の問題を自動的に解決しません。正確な結果を確実に得るためには、修復が適用された後に再度スキャンする必要があります。

5.6.5.10. 修復の適用解除

以前に適用された修復を適用解除することが必要になる場合があります。

手順

1. **apply** フラグを **false** に設定します。

```
$ oc -n openshift-compliance \
patch complianceremediations/rhcos4-moderate-worker-sysctl-net-ipv4-conf-all-accept-redirects \
--patch '{"spec":{"apply":false}}' --type=merge
```

2. 修復ステータスは **NotApplied** に変更され、複合の **MachineConfig** オブジェクトは修復を含まないように再度レンダリングされます。

重要

修復を含む影響を受けるすべてのノードが再起動されます。

重要

Compliance Operator は、修復の間に発生する可能性のある依存関係の問題を自動的に解決しません。正確な結果を確実に得るためには、修復が適用された後に再度スキャンする必要があります。

5.6.5.11. KubeletConfig 修復の削除

KubeletConfig の修正は、ノードレベルのプロファイルに含まれています。KubeletConfig 修復を削除するには、**KubeletConfig** オブジェクトから手動で削除する必要があります。この例は、**one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 修正のコンプライアンスチェックを削除する方法を示しています。

手順

1. **one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available** 修復の **scan-name** およびコンプライアンスチェックを見つけます。

```
$ oc -n openshift-compliance get remediation \ one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available -o yaml
```

出力例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceRemediation
metadata:
  annotations:
    compliance.openshift.io/xccdf-value-used: var-kubelet-evictionhard-imagefs-available
  creationTimestamp: "2022-01-05T19:52:27Z"
  generation: 1
  labels:
    compliance.openshift.io/scan-name: one-rule-tp-node-master 1
    compliance.openshift.io/suite: one-rule-ssb-node
  name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
  namespace: openshift-compliance
  ownerReferences:
  - apiVersion: compliance.openshift.io/v1alpha1
    blockOwnerDeletion: true
    controller: true
    kind: ComplianceCheckResult
    name: one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available
    uid: fe8e1577-9060-4c59-95b2-3e2c51709adc
  resourceVersion: "84820"
  uid: 5339d21a-24d7-40cb-84d2-7a2ebb015355
spec:
  apply: true
  current:
    object:
      apiVersion: machineconfiguration.openshift.io/v1
      kind: KubeletConfig
```

```
      spec:
        kubeletConfig:
          evictionHard:
            imagefs.available: 10% (2)
  outdated: {}
  type: Configuration
status:
  applicationState: Applied
```

1. 修復のスキャン名。
2. **KubeletConfig** オブジェクトに追加された修復。

注記

修復によって **evictionHard** kubelet 設定が呼び出される場合は、すべての **evictionHard** パラメーター (**memory.available**、**nodefs.available**、**nodefs.inodesFree**、**imagefs.available**、および **imagefs.inodesFree**) を指定する必要があります。すべてのパラメーターを指定しないと、指定したパラメーターのみが適用され、修正が正しく機能しません。

2. 修復を削除します。

   a. 修復オブジェクトの **apply** を false に設定します。

```
$ oc -n openshift-compliance patch \
complianceremediations/one-rule-tp-node-master-kubelet-eviction-thresholds-set-hard-imagefs-available \
-p '{"spec":{"apply":false}}' --type=merge
```

   b. **scan-name** を使用して、修復が適用された **KubeletConfig** オブジェクトを見つけます。

```
$ oc -n openshift-compliance get kubeletconfig \
--selector compliance.openshift.io/scan-name=one-rule-tp-node-master
```

出力例

```
NAME                                  AGE
compliance-operator-kubelet-master   2m34s
```

   c. **KubeletConfig** オブジェクトから修復 **imagefs.available: 10%** を手動で削除します。

```
$ oc edit -n openshift-compliance KubeletConfig compliance-operator-kubelet-master
```

重要

修復を含む影響を受けるすべてのノードが再起動されます。

注記

また、修正を自動適用する調整済みプロファイルのスケジュールされたスキャンからルールを除外する必要があります。除外しない場合、修正は次のスケジュールされたスキャン中に再適用されます。

5.6.5.12. 一貫性のない ComplianceScan

ScanSetting オブジェクトは、**ScanSetting** または **ScanSettingBinding** オブジェクトから生成されるコンプライアンススキャンがスキャンするノードロールを一覧表示します。通常、各ノードのロールはマシン設定プールにマップされます。

重要

マシン設定プールのすべてのマシンが同一であり、プールのノードからのすべてのスキャン結果が同一であることが予想されます。

一部の結果が他とは異なる場合、Compliance Operator は一部のノードが **INCONSISTENT** として報告される **ComplianceCheckResult** オブジェクトにフラグを付けます。すべての **ComplianceCheckResult** オブジェクトには、**compliance.openshift.io/inconsistent-check** のラベルも付けられます。

プール内のマシン数は非常に大きくなる可能性があるため、Compliance Operator は最も一般的な状態を検索し、一般的な状態とは異なるノードを一覧表示しようとします。最も一般的な状態は **compliance.openshift.io/most-common-status** アノテーションに保存され、アノテーション **compliance.openshift.io/inconsistent-source** には、最も一般的なステータスとは異なるチェックステータスの **hostname:status** のペアが含まれます。共通状態が見つからない場合、すべての **hostname:status** ペアが **compliance.openshift.io/inconsistent-source** アノテーションにリスト表示されます。

可能な場合は、修復が依然として作成され、クラスターが準拠したステータスに収束できるようにします。ただし、これが常に実行可能な訳ではなく、ノード間の差異の修正は手作業で行う必要があります。スキャンに **compliance.openshift.io/rescan=** オプションのアノテーションを付けて一貫性のある結果を取得するために、コンプライアンススキャンを再実行する必要があります。

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

5.6.5.13. 関連情報

- Modifying nodes

5.6.6. 高度な Compliance Operator タスクの実行

Compliance Operator は、デバッグや既存のツールとの統合を目的とした上級ユーザー向けのオプションを備えています。

5.6.6.1. ComplianceSuite オブジェクトおよび ComplianceScan オブジェクトの直接使用

ユーザーは **ScanSetting** および **ScanSettingBinding** オブジェクトを利用してスイートおよびスキャンを定義することが推奨されますが、**ComplianceSuite** オブジェクトを直接定義する有効なユースケースもあります。

- スキャンする単一ルールのみを指定する。これは、デバッグモードが非常に詳細な情報を取得する可能性があるため、**debug: true** 属性を使用してデバッグする際に役立ちます。これにより、OpenSCAP スキャナーの詳細度が上がります。テストを 1 つのルールに制限すると、デバッグ情報の量を減らすことができます。

- カスタム nodeSelector を指定する。修復を適用可能にするには、nodeSelector はプールと一致する必要があります。

- テーラリングファイルでスキャンをカスタム設定マップをポイントする。

- テストまたは開発目的の場合で、バンドルからのプロファイルの解析に伴うオーバーヘッドが不要な場合。

以下の例は、単一のルールのみでワーカーマシンをスキャンする **ComplianceSuite** を示しています。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-rhel8@sha256:45dc...
      debug: true
      rule: xccdf_org.ssgproject.content_rule_no_direct_root_logins
      nodeSelector:
      node-role.kubernetes.io/worker: ""
```

上記の **ComplianceSuite** オブジェクトおよび **ComplianceScan** オブジェクトは、OpenSCAP が想定する形式で複数の属性を指定します。

プロファイル、コンテンツ、またはルールの値を見つけるには、最初に **ScanSetting** および **ScanSettingBinding** から同様の Suite を作成して開始するか、ルールやプロファイルなどの **ProfileBundle** オブジェクトから解析されたオブジェクトを検査することができます。これらのオブジェクトには、**ComplianceSuite** から参照するために使用できる **xccdf_org** 識別子が含まれます。

5.6.6.2. ScanSetting スキャンの PriorityClass の設定

大規模な環境では、デフォルトの **PriorityClass** オブジェクトの優先順位が低すぎて、Pod が時間どおりにスキャンを実行することを保証できない場合があります。コンプライアンスを維持するか、自動スキャンを保証する必要のあるクラスターの場合、**PriorityClass** 変数を設定して、Compliance Operator がリソースの制約の状況で常に優先順位が指定されるようにします。

前提条件

- オプション: **PriorityClass** オブジェクトを作成した。詳細は、**関連情報** の「優先順位およびプリエンプションの設定」を参照してください。

手順

- **PriorityClass** 変数を設定します。

```
apiVersion: compliance.openshift.io/v1alpha1
strictNodeScan: true
metadata:
  name: default
  namespace: openshift-compliance
priorityClass: compliance-high-priority 1
kind: ScanSetting
showNotApplicable: false
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ''
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    - effect: NoSchedule
      key: node.kubernetes.io/memory-pressure
      operator: Exists
schedule: 0 1 * * *
roles:
  - master
  - worker
scanTolerations:
  - operator: Exists
```

1. **ScanSetting** で参照される **PriorityClass** が見つからない場合、Operator は **PriorityClass** を空のままで警告を発行し、**PriorityClass** なしでスキャンのスケジュールを続行します。

関連情報

- 優先順位およびプリエンプションの設定

5.6.6.3. 未加工の調整済みプロファイルの使用

TailoredProfile CR は最も一般的なテーラリング操作を有効にする一方で、XCCDF 標準は OpenSCAP プロファイルの調整におけるより多くの柔軟性を提供します。さらに、組織が以前に OpenScap を使用していた場合、既存の XCCDF テーラリングファイルが存在し、これを再利用できる可能性があります。

ComplianceSuite オブジェクトには、カスタムのテーラリングファイルにポイントできるオプションの **TailoringConfigMap** 属性が含まれます。**TailoringConfigMap** 属性の値は設定マップの名前です。

これには、**tailoring.xml** というキーが含まれる必要があり、このキーの値はテーラリングのコンテンツです。

手順

1. ファイルから **ConfigMap** オブジェクトを作成します。

```
$ oc -n openshift-compliance \
create configmap nist-moderate-modified \
--from-file=tailoring.xml=/path/to/the/tailoringFile.xml
```

2. スイートに属するスキャンでテーラリングファイルを参照します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ComplianceSuite
metadata:
  name: workers-compliancesuite
spec:
  debug: true
  scans:
    - name: workers-scan
      profile: xccdf_org.ssgproject.content_profile_moderate
      content: ssg-rhcos4-ds.xml
      contentImage: registry.redhat.io/compliance/openshift-compliance-content-
rhel8@sha256:45dc...
      debug: true
  tailoringConfigMap:
      name: nist-moderate-modified
  nodeSelector:
    node-role.kubernetes.io/worker: ""
```

5.6.6.4. 再スキャンの実行

通常、毎週月曜日または日次など、定義されたスケジュールでスキャンを再実行する必要がある場合があります。また、ノードの問題を修正した後にスキャンを 1 回再実行することは役に立ちます。単一スキャンを実行するには、スキャンに **compliance.openshift.io/rescan=** オプションでアノテーションを付けます。

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

再スキャンにより、**rhcos-moderate** プロファイルの 4 つの追加 **mc** が生成されます。

```
$ oc get mc
```

出力例

```
75-worker-scan-chronyd-or-ntpd-specify-remote-server
75-worker-scan-configure-usbguard-auditbackend
75-worker-scan-service-usbguard-enabled
75-worker-scan-usbguard-allow-hid-and-hub
```

重要

スキャン設定の **default-auto-apply** ラベルが適用されると、修復は自動的に適用され、古い修復が自動的に更新されます。依存関係により適用されなかった修復や、古くなった修復がある場合には、再スキャンにより修復が適用され、再起動がトリガーされる可能性があります。**MachineConfig** オブジェクトを使用する修復のみが再起動をトリガーします。適用する更新または依存関係がない場合は、再起動は発生しません。

5.6.6.5. 結果に関するカスタムストレージサイズの設定

ComplianceCheckResult などのカスタムリソースは、スキャンされたすべてのノード間での単一チェックの集計された結果を表しますが、スキャナーによって生成される未加工の結果を確認することが役に立つ場合もあります。未加工の結果は ARF 形式で生成され、サイズが大きくなる可能性がありますが (ノードあたり数十メガバイト)、それらを **etcd** のキーと値のストアでサポートされる Kubernetes リソースに保存することは実際的ではありません。すべてのスキャンは、1GB のサイズにデフォルト設定される永続ボリューム (PV) を作成します。環境によっては、PV サイズを適宜増やす必要がある場合があります。これは、**ScanSetting** および **ComplianceScan** リソースの両方に公開される **rawResultStorage.size** 属性を使用して実行されます。

関連するパラメーターは **rawResultStorage.rotation** であり、これは古いスキャンのローテーションが行われる前に保持されるスキャンの数を制御します。デフォルト値は 3 です。ローテーションポリシーを 0 に設定するとローテーションは無効になります。デフォルトのローテーションポリシーと、未加工の ARF スキャンレポートにつき 100 MB を見積もり、お使いの環境に適した PV サイズを計算できます。

5.6.6.5.1. カスタム結果ストレージ値の使用

OpenShift Container Platform は各種のパブリッククラウドまたはベアメタルにデプロイできるので、Compliance Operator は利用可能なストレージ設定を判別できません。デフォルトで、Compliance Operator はクラスターのデフォルトのストレージクラスを使用して結果を保存するために PV の作成を試行しますが、カスタムストレージクラスは **rawResultStorage.StorageClassName** 属性を使用して設定できます。

重要

クラスターがデフォルトのストレージクラスを指定しない場合、この属性を設定する必要があります。

標準ストレージクラスを使用するように **ScanSetting** カスタムリソースを設定し、サイズが 10GB の永続ボリュームを作成し、最後の 10 件の結果を保持します。

ScanSetting CR の例

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  storageClassName: standard
  rotation: 10
  size: 10Gi
roles:
- worker
```

```
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
```

5.6.6.6. スイートスキャンによって生成される修復の適用

ComplianceSuite オブジェクトで **autoApplyRemediations** ブール値パラメーターを使用することもできますが、オブジェクトに **compliance.openshift.io/apply-remediations** のアノテーションを付けることもできます。これにより、Operator は作成されるすべての修復を適用できます。

手順

- 以下を実行して、**compliance.openshift.io/apply-remediations** アノテーションを適用します。

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/apply-remediations=
```

5.6.6.7. 修復の自動更新

新しいコンテンツを含むスキャンは、修復に **OUTDATED** というマークを付ける可能性があります。管理者は、**compliance.openshift.io/remove-outdated** アノテーションを適用して新規の修復を適用し、古い修復を削除できます。

手順

- **compliance.openshift.io/remove-outdated** アノテーションを適用します。

```
$ oc -n openshift-compliance \
annotate compliancesuites/workers-compliancesuite compliance.openshift.io/remove-outdated=
```

または、**ScanSetting** または **ComplianceSuite** オブジェクトに **autoUpdateRemediations** フラグを設定し、修復を自動的に更新します。

5.6.6.8. Compliance Operator 用のカスタム SCC の作成

環境によっては、Compliance Operator の **api-resource-collector** に適切な権限が確実に提供されるように、カスタムの Security Context Constraints (SCC) ファイルを作成する必要があります。

前提条件

- **admin** 権限がある。

手順

1. **restricted-adjusted-compliance.yaml** という名前の YAML ファイルで SCC を定義します。

   SecurityContextConstraints オブジェクト定義

```yaml
allowHostDirVolumePlugin: false
allowHostIPC: false
allowHostNetwork: false
allowHostPID: false
allowHostPorts: false
allowPrivilegeEscalation: true
allowPrivilegedContainer: false
allowedCapabilities: null
apiVersion: security.openshift.io/v1
defaultAddCapabilities: null
fsGroup:
  type: MustRunAs
kind: SecurityContextConstraints
metadata:
  name: restricted-adjusted-compliance
priority: 30 1
readOnlyRootFilesystem: false
requiredDropCapabilities:
- KILL
- SETUID
- SETGID
- MKNOD
runAsUser:
  type: MustRunAsRange
seLinuxContext:
  type: MustRunAs
supplementalGroups:
  type: RunAsAny
users:
- system:serviceaccount:openshift-compliance:api-resource-collector 2
volumes:
- configMap
- downwardAPI
- emptyDir
- persistentVolumeClaim
- projected
- secret
```

1. この SCC の優先度は、**system:authenticated** グループに適用される他のどの SCC よりも高くなければなりません。
2. Compliance Operator スキャナー Pod によって使用されるサービスアカウント。

2. SCC を作成します。

```
$ oc create -n openshift-compliance  -f restricted-adjusted-compliance.yaml
```

出力例

```
securitycontextconstraints.security.openshift.io/restricted-adjusted-compliance created
```

検証

1. SCC が作成されたことを確認します。

```
$ oc get -n openshift-compliance scc restricted-adjusted-compliance
```

出力例

```
NAME                    PRIV   CAPS      SELINUX   RUNASUSER      FSGROUP
SUPGROUP   PRIORITY   READONLYROOTFS   VOLUMES
restricted-adjusted-compliance   false   <no value>   MustRunAs   MustRunAsRange
MustRunAs   RunAsAny   30        false
["configMap","downwardAPI","emptyDir","persistentVolumeClaim","projected","secret"]
```

5.6.6.9. 関連情報

- Security Context Constraints の管理

5.6.7. Compliance Operator スキャンのトラブルシューティング

このセクションでは、Compliance Operator のトラブルシューティングの方法を説明します。この情報は、問題を診断したり、バグレポートに情報を提供したりする際に役立ちます。一般的なヒントには、以下が含まれます。

- Compliance Operator は、重要なことが発生すると Kubernetes イベントを生成します。コマンドを使用して、クラスター内のすべてのイベントを表示できます。

  ```
  $ oc get events -n openshift-compliance
  ```

  または、コマンドを使用してスキャンなどのオブジェクトのイベントを表示します。

  ```
  $ oc describe -n openshift-compliance compliancescan/cis-compliance
  ```

- Compliance Operator は複数のコントローラーで構成されており、API オブジェクトごとに約 1 つのコントローラーで設定されます。問題のある API オブジェクトに対応するコントローラーのみをフィルターすることが役に立つ場合があります。**ComplianceRemediation** を適用できない場合、**remediationctrl** コントローラーのメッセージを表示します。**jq** で解析することにより、単一のコントローラーからのメッセージをフィルターできます。

  ```
  $ oc -n openshift-compliance logs compliance-operator-775d7bddbd-gj58f \
      | jq -c 'select(.logger == "profilebundlectrl")'
  ```

- タイムスタンプには、UTC の UNIX エポックからの経過時間で秒単位でログに記録されます。人間が判読可能な日付に変換するには、**date -d @timestamp --utc** を使用します。以下は例になります。

  ```
  $ date -d @1596184628.955853 --utc
  ```

- 数多くのカスタムリソースで、最も重要な **ComplianceSuite** および **ScanSetting** で **debug** オプションを設定できます。このオプションを有効にすると、OpenSCAP スキャナー Pod や他のヘルパー Pod の詳細度が上がります。

- 単一ルールの合否が予期せずに出される場合、そのルールのみを使用して単一スキャンまたはスイートを実行し、対応する **ComplianceCheckResult** オブジェクトからルール ID を見つ

け、それを **Scan** CR の **rule** 属性として使用することが役に立つ場合があります。次に、**debug** オプションが有効になると、スキャナー Pod の **scanner** コンテナーログに未加工の OpenSCAP ログが表示されます。

5.6.7.1. スキャンの仕組み

以下のセクションでは、Compliance Operator スキャンのコンポーネントおよびステージを説明します。

5.6.7.1.1. コンプライアンスのソース

コンプライアンスコンテンツは、**ProfileBundle** オブジェクトから生成される **Profile** オブジェクトに保存されます。Compliance Operator は、**ProfileBundle** をクラスター用とクラスターノード用に作成します。

```
$ oc get -n openshift-compliance profilebundle.compliance
```

```
$ oc get -n openshift-compliance profile.compliance
```

ProfileBundle オブジェクトは、**Bundle** の名前でラベルが付けられたデプロイメントで処理されます。**Bundle** で問題のトラブルシューティングを行うには、デプロイメントを見つけ、デプロイメントで Pod のログを表示できます。

```
$ oc logs -n openshift-compliance -lprofile-bundle=ocp4 -c profileparser
```

```
$ oc get -n openshift-compliance deployments,pods -lprofile-bundle=ocp4
```

```
$ oc logs -n openshift-compliance pods/<pod-name>
```

```
$ oc describe -n openshift-compliance pod/<pod-name> -c profileparser
```

5.6.7.1.2. ScanSetting および ScanSettingBinding のライフサイクルおよびデバッグ

有効なコンプライアンスコンテンツソースを使用して、高レベルの **ScanSetting** および **ScanSettingBinding** オブジェクトを、**ComplianceSuite** および **ComplianceScan** オブジェクトを生成するために使用できます。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: my-companys-constraints
debug: true
# For each role, a separate scan will be created pointing
# to a node-role specified in roles
roles:
  - worker
---
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSettingBinding
metadata:
  name: my-companys-compliance-requirements
profiles:
```

```
    # Node checks
    - name: rhcos4-e8
      kind: Profile
      apiGroup: compliance.openshift.io/v1alpha1
    # Cluster checks
    - name: ocp4-e8
      kind: Profile
      apiGroup: compliance.openshift.io/v1alpha1
  settingsRef:
    name: my-companys-constraints
    kind: ScanSetting
    apiGroup: compliance.openshift.io/v1alpha1
```

ScanSetting および **ScanSettingBinding** オブジェクトはどちらも、**logger=scansettingbindingctrl** のタグの付けられた同じコントローラーで処理されます。これらのオブジェクトにはステータスがありません。問題はイベントの形式で通信されます。

```
Events:
  Type     Reason        Age    From                    Message
  ----     ------        ----   ----                    -------
  Normal   SuiteCreated  9m52s  scansettingbindingctrl  ComplianceSuite openshift-compliance/my-companys-compliance-requirements created
```

今回のリリースにより、**ComplianceSuite** オブジェクトが作成されました。フローは新規に作成された **ComplianceSuite** を継続して調整します。

5.6.7.1.3. ComplianceSuite カスタムリソースのライフサイクルおよびデバッグ

ComplianceSuite CR は **ComplianceScan** CR に関連したラッパーです。**ComplianceSuite** CR は、**logger=suitectrl** のタグが付けられたコントローラーによって処理されます。このコントローラーは、スイートからのスキャンの作成を処理し、個別のスキャンのステータスを調整し、これを単一の Suite ステータスに集約します。スイートが定期的に実行するよう設定されている場合、**suitectrl** は、初回の実行後にスキャンをスイートで再実行する **CronJob** CR の作成にも対応します。

```
$ oc get cronjobs
```

出力例

```
NAME                                SCHEDULE    SUSPEND   ACTIVE   LAST SCHEDULE   AGE
<cron_name>                         0 1 * * *   False     0        <none>          151m
```

最も重要な問題について、イベントが生成されます。**oc describe compliancesuites/<name>** を使用してそれらを表示します。**Suite** オブジェクトには、**Status** サブリソースも含まれており、これはこのスイートに属する **Scan** オブジェクトが **Status** サブリソースを更新すると更新されます。予想されるすべてのスキャンが作成されると、コントローラーがスキャンコントローラーに渡されます。

5.6.7.1.4. ComplianceScan カスタムリソースのライフサイクルおよびデバッグ

ComplianceScan CR は **scanctrl** コントローラーによって処理されます。これは、実際のスキャンとスキャン結果が作成される場所でもあります。それぞれのスキャンは複数のフェーズを経由します。

5.6.7.1.4.1. Pending (保留) フェーズ

このフェーズでは、スキャンがその正確性を検証します。ストレージサイズなどの一部のパラメーターが無効な場合、スキャンは ERROR 結果と共に DONE に移行します。それ以外の場合は、Launching (起動) フェーズに進みます。

5.6.7.1.4.2. Launching (起動) フェーズ

このフェーズでは、いくつかの config map にスキャナー Pod の環境またはスキャナー Pod が評価するスクリプトが直接含まれます。config map をリスト表示します。

```
$ oc -n openshift-compliance get cm \
-l compliance.openshift.io/scan-name=rhcos4-e8-worker,complianceoperator.openshift.io/scan-script=
```

これらの config map はスキャナー Pod によって使用されます。スキャナーの動作を変更したり、スキャナーのデバッグレベルを変更したり、未加工の結果を出力したりする必要がある場合は、1 つの方法として config map を変更することができます。その後、未加工の ARF の結果を保存するために永続ボリューム要求がスキャンごとに作成されます。

```
$ oc get pvc -n openshift-compliance -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

PVC はスキャンごとの **ResultServer** デプロイメントでマウントされます。**ResultServer** は、個別のスキャナー Pod が完全な ARF 結果をアップロードする単純な HTTP サーバーです。各サーバーは、異なるノードで実行できます。完全な ARF の結果のサイズは非常に大きくなる可能性があり、複数のノードから同時にマウントできるボリュームを作成することができると想定することはできません。スキャンが終了した後に、**ResultServer** デプロイメントはスケールダウンされます。未加工の結果のある PVC は別のカスタム Pod からマウントでき、結果はフェッチしたり、検査したりできます。スキャナー Pod と **ResultServer** 間のトラフィックは相互 TLS プロトコルで保護されます。

最後に、スキャナー Pod はこのフェーズで起動します。**Platform** スキャンインスタンスの 1 つのスキャナー Pod と、**node** スキャンインスタンスの一致するノードごとに 1 つのスキャナー Pod です。ノードごとの Pod にはノード名のラベルが付けられます。それぞれの Pod には、常に **ComplianceScan** という名前のラベルが付けられます。

```
$ oc get pods -lcompliance.openshift.io/scan-name=rhcos4-e8-worker,workload=scanner --show-labels
```

出力例

```
NAME                                                        READY   STATUS      RESTARTS   AGE   LABELS
rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod   0/2     Completed   0          39m
compliance.openshift.io/scan-name=rhcos4-e8-worker,targetNode=ip-10-0-169-90.eu-north-1.compute.internal,workload=scanner
```

+ スキャンは Running (実行) フェーズに進みます。

5.6.7.1.4.3. Running (実行) フェーズ

実行フェーズは、スキャナー Pod の完了後に開始します。以下の用語およびプロセスは実行フェーズで使用されます。

- **init container**: **content-container** という 1 つの init コンテナーがあります。これは、**contentImage** コンテナーを実行し、**contentFile** を、この Pod の他のコンテナーで共有される **/content** ディレクトリーにコピーします。

- scanner: このコンテナーはスキャンを実行します。ノードのスキャンの場合には、コンテナーはノードファイルシステムを **/host** としてマウントし、init コンテナーによって配信されるコンテンツをマウントします。また、コンテナーは Launching (起動) フェーズで作成される **entrypoint ConfigMap** をマウントし、これを実行します。エントリーポイント **ConfigMap** のデフォルトスクリプトは OpenSCAP を実行し、結果ファイルを Pod のコンテナー間で共有される **/results** ディレクトリーに保存します。この Pod のログを表示して、OpenSCAP スキャナーがチェックした内容を判別できます。より詳細な出力は、**debug** フラグを使用して表示できます。

- logcollector: logcollector コンテナーは、スキャナーコンテナーが終了するまで待機します。その後、これは **ConfigMap** として完全な ARF 結果を **ResultServer** にアップロードし、スキャン結果および OpenSCAP 結果コードと共に XCCDF 結果を個別にアップロードします。これらの結果の config map には、スキャン名 (**compliance.openshift.io/scan-name=rhcos4-e8-worker**) のラベルが付けられます。

```
$ oc describe cm/rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
```

出力例

```
Name:         rhcos4-e8-worker-ip-10-0-169-90.eu-north-1.compute.internal-pod
Namespace:    openshift-compliance
Labels:       compliance.openshift.io/scan-name-scan=rhcos4-e8-worker
              complianceoperator.openshift.io/scan-result=
Annotations:  compliance-remediations/processed:
              compliance.openshift.io/scan-error-msg:
              compliance.openshift.io/scan-result: NON-COMPLIANT
              OpenSCAP-scan-result/node: ip-10-0-169-90.eu-north-1.compute.internal

Data
====
exit-code:
----
2
results:
----
<?xml version="1.0" encoding="UTF-8"?>
...
```

Platform スキャンのスキャナー Pod も同様ですが、以下の点で異なります。

- **api-resource-collector** という追加の init コンテナーがあります。これは content-container init、コンテナーで提供される OpenSCAP コンテンツを読み取り、コンテンツで確認する必要のある API リソースを判別し、それらの API リソースを **scanner** コンテナーが読み取りを行う共有ディレクトリーに保存します。

- **scanner** コンテナーは、ホストファイルシステムをマウントする必要はありません。

スキャナー Pod が実行されると、スキャンは Aggregating (集計) フェーズに移行します。

5.6.7.1.4.4. Aggregating (集計) フェーズ

Aggregating (集計) フェーズでは、スキャンコントローラーがアグリゲーター Pod と呼ばれる別の Pod を起動します。その目的は、結果の **ConfigMap** オブジェクトを取り、結果を読み取り、それぞれのチェックの結果に対応する Kubernetes オブジェクトを作成することにあります。チェックの失敗を自

動的に修復できる場合は、**ComplianceRemediation** オブジェクトが作成されます。チェックと修復について人間が判読できるメタデータを提供するために、アグリゲーター Pod は init コンテナーを使用して OpenSCAP コンテンツもマウントします。

設定マップがアグリゲーター Pod によって処理される場合、これには **compliance-remediations/processed** ラベルでラベルが付けられます。このフェーズの結果は **ComplianceCheckResult** オブジェクトになります。

```
$ oc get compliancecheckresults -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

出力例

```
NAME                                              STATUS   SEVERITY
rhcos4-e8-worker-accounts-no-uid-except-zero           PASS     high
rhcos4-e8-worker-audit-rules-dac-modification-chmod       FAIL     medium
```

ComplianceRemediation オブジェクト:

```
$ oc get complianceremediations -lcompliance.openshift.io/scan-name=rhcos4-e8-worker
```

出力例

```
NAME                                              STATE
rhcos4-e8-worker-audit-rules-dac-modification-chmod       NotApplied
rhcos4-e8-worker-audit-rules-dac-modification-chown       NotApplied
rhcos4-e8-worker-audit-rules-execution-chcon            NotApplied
rhcos4-e8-worker-audit-rules-execution-restorecon        NotApplied
rhcos4-e8-worker-audit-rules-execution-semanage          NotApplied
rhcos4-e8-worker-audit-rules-execution-setfiles          NotApplied
```

これらの CR が作成されると、アグリゲーター Pod は終了し、スキャンは Done (終了) フェーズに移行します。

5.6.7.1.4.5. Done (終了) フェーズ

最終のスキャンフェーズでは、必要な場合にスキャンリソースがクリーンアップされ、**ResultServer** デプロイメントは (スキャンが 1 回限りの場合) スケールダウンされるか、(スキャンが継続される場合) 削除されます。次回のスキャンインスタンスではデプロイメントを再作成します。

また、Done (終了) フェーズでは、スキャンにアノテーションを付けてスキャンの再実行をトリガーすることもできます。

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

スキャンが Done (終了) フェーズに到達した後に、修復が **autoApplyRemediations: true** を指定して自動的に適用されるように設定されていない限り、自動的に実行されることは何もありません。OpenShift Container Platform 管理者は、修復を確認し、必要に応じてそれらを適用できるようになりました。修復が自動的に適用されるように設定されている場合、**ComplianceSuite** コントローラーが Done (終了) フェーズで引き継ぎ、マシン設定プールをスキャンがマップされるポイントで一時停止し、すべての修復を 1 回で適用します。修復が適用されると、**ComplianceRemediation** コントローラーが引き継ぎます。

5.6.7.1.5. ComplianceRemediation コントローラーのライフサイクルおよびデバッグ

サンプルスキャンは特定の結果を報告します。修復の 1 つを有効にするには、**apply** 属性を **true** に切り替えます。

```
$ oc patch complianceremediations/rhcos4-e8-worker-audit-rules-dac-modification-chmod --patch
'{"spec":{"apply":true}}' --type=merge
```

ComplianceRemediation コントローラー (**logger=remediationctrl**) は変更されたオブジェクトを調整します。調整の結果として、調整される修復オブジェクトのステータスが変更されますが、適用されたすべての修復が含まれる、レンダリングされるスイートごとの **MachineConfig** オブジェクトも変更されます。

MachineConfig オブジェトは常に **75-** で開始し、スキャンとスィートに基づいて名前が付けられます。

```
$ oc get mc | grep 75-
```

出力例

```
75-rhcos4-e8-worker-my-companys-compliance-requirements                         3.2.0
2m46s
```

現在 **mc** を構成する修復は、マシン設定のアノテーションにリスト表示されます。

```
$ oc describe mc/75-rhcos4-e8-worker-my-companys-compliance-requirements
```

出力例

```
Name:        75-rhcos4-e8-worker-my-companys-compliance-requirements
Labels:      machineconfiguration.openshift.io/role=worker
Annotations:  remediation/rhcos4-e8-worker-audit-rules-dac-modification-chmod:
```

ComplianceRemediation コントローラーのアルゴリズムは以下のようになります。

- 現在適用されているすべての修復は初期の修復セットに読み込まれます。
- 調整された修復が適用されることが予想される場合、それはセットに追加されます。
- **MachineConfig** オブジェクトはセットからレンダリングされ、セット内の修復の名前でアノテーションが付けられます。セットが空の場合 (最後の修復は適用されない)、レンダリングされる **MachineConfig** オブジェクトは削除されます。
- レンダリングされたマシン設定がクラスターにすでに適用されているものとは異なる場合にのみ、適用される MC は更新されます (または作成されるか、削除されます)。
- **MachineConfig** オブジェクトの作成または変更により、**machineconfiguration.openshift.io/role** ラベルに一致するノードの再起動がトリガーされます。詳細は、Machine Config Operator のドキュメントを参照してください。

修復ループは、レンダリングされたマシン設定が更新され (必要な場合)、調整された修復オブジェクトのステータスが更新されると終了します。この場合、修復を適用すると再起動がトリガーされます。再起動後、スキャンにアノテーションを付け、再度実行します。

```
$ oc -n openshift-compliance \
annotate compliancescans/rhcos4-e8-worker compliance.openshift.io/rescan=
```

スキャンが実行され、終了します。渡される修復の有無を確認します。

```
$ oc -n openshift-compliance \
get compliancecheckresults/rhcos4-e8-worker-audit-rules-dac-modification-chmod
```

出力例

```
NAME                                                 STATUS   SEVERITY
rhcos4-e8-worker-audit-rules-dac-modification-chmod   PASS     medium
```

5.6.7.1.6. 役に立つラベル

Compliance Operator によって起動する各 Pod には、それが属するスキャンおよびその機能にとくに関連するラベルが付けられます。スキャン ID には **compliance.openshift.io/scan-name** ラベルが付けられます。ワークロード ID には、**workload** ラベルでラベルが付けられます。

Compliance Operator は以下のワークロードをスケジュールします。

- **scanner**: コンプライアンススキャンを実行します。
- **resultserver**: コンプライアンススキャンの未加工の結果を保存します。
- **aggregator**: 結果を集計し、不整合を検出し、結果オブジェクト (チェックの結果と修復) を出力します。
- **suitererunner**: 再実行するスイートにタグを付けます (スケジュールが設定されている場合)。
- **profileparser**: データストリームを解析し、適切なプロファイル、ルールおよび変数を作成します。

デバッグおよびログが特定のワークロードに必要な場合は、以下を実行します。

```
$ oc logs -l workload=<workload_name> -c <container_name>
```

5.6.7.2. Compliance Operator のリソース制限の引き上げ

Compliance Operator は、デフォルトの制限よりもメモリーを多く必要とする場合があります。この問題を軽減する最善の方法は、カスタムリソースの制限を設定することです。

スキャナー Pod のデフォルトのメモリーおよび CPU 制限を増やすには、**ScanSetting カスタムリソース** を参照してください。

手順

1. Operator のメモリー制限を 500 Mi に増やすには、**co-memlimit-patch.yaml** という名前の以下のパッチファイルを作成します。

```
spec:
  config:
    resources:
```

```
    limits:
      memory: 500Mi
```

2. パッチファイルを適用します。

```
$ oc patch sub compliance-operator -nopenshift-compliance --patch-file co-memlimit-patch.yaml --type=merge
```

5.6.7.3. Operator リソース制約の設定

resources フィールドは、Operator Lifecycle Manager (OLM) によって作成される Pod 内のすべてのコンテナーの Resource Constraints を定義します。

注記

このプロセスで適用されるリソース制約は、既存のリソース制約を上書きします。

手順

- **Subscription** オブジェクトを編集して、各コンテナーに 0.25 cpu と 64 Mi のメモリーの要求と、0.5 cpu と 128 Mi のメモリーの制限を挿入します。

```
kind: Subscription
metadata:
  name: compliance-operator
  namespace: openshift-compliance
spec:
  package: package-name
  channel: stable
  config:
    resources:
      requests:
        memory: "64Mi"
        cpu: "250m"
      limits:
        memory: "128Mi"
        cpu: "500m"
```

5.6.7.4. ScanSetting リソースの設定

500 を超える MachineConfig を含むクラスターで Compliance Operator を使用する場合、**Platform** スキャンを実行するときに、**ocp4-pci-dss-api-checks-pod** Pod が **init** フェーズで一時停止することがあります。

注記

このプロセスで適用されるリソース制約は、既存のリソース制約を上書きします。

手順

1. **ocp4-pci-dss-api-checks-pod** Pod が **Init:OOMKilled** ステータスでスタックしていることを確認します。

```
$ oc get pod ocp4-pci-dss-api-checks-pod -w
```

出力例

```
NAME                        READY   STATUS        RESTARTS        AGE
ocp4-pci-dss-api-checks-pod   0/2     Init:1/2      8 (5m56s ago)   25m
ocp4-pci-dss-api-checks-pod   0/2     Init:OOMKilled   8 (6m19s ago)   26m
```

2. **ScanSetting** CR の **scanLimits** 属性を編集して、**ocp4-pci-dss-api-checks-pod** Pod で使用可能なメモリーを増やします。

```yaml
timeout: 30m
strictNodeScan: true
metadata:
  name: default
  namespace: openshift-compliance
kind: ScanSetting
showNotApplicable: false
rawResultStorage:
  nodeSelector:
    node-role.kubernetes.io/master: ''
  pvAccessModes:
    - ReadWriteOnce
  rotation: 3
  size: 1Gi
  tolerations:
    - effect: NoSchedule
      key: node-role.kubernetes.io/master
      operator: Exists
    - effect: NoExecute
      key: node.kubernetes.io/not-ready
      operator: Exists
      tolerationSeconds: 300
    - effect: NoExecute
      key: node.kubernetes.io/unreachable
      operator: Exists
      tolerationSeconds: 300
    - effect: NoSchedule
      key: node.kubernetes.io/memory-pressure
      operator: Exists
schedule: 0 1 * * *
roles:
  - master
  - worker
apiVersion: compliance.openshift.io/v1alpha1
maxRetryOnTimeout: 3
scanTolerations:
  - operator: Exists
scanLimits:
  memory: 1024Mi 1
```

1 デフォルト設定は **500Mi** です。

3. **ScanSetting** CR をクラスターに適用します。

```
$ oc apply -f scansetting.yaml
```

5.6.7.5. ScanSetting タイムアウトの設定

ScanSetting オブジェクトには、**ComplianceScanSetting** オブジェクトで **1h30m** などの期間文字列として指定できるタイムアウトオプションがあります。指定されたタイムアウト内にスキャンが終了しない場合、スキャンは **maxRetryOnTimeout** 制限に達するまで再試行されます。

手順

- ScanSetting で **timeout** と **maxRetryOnTimeout** を設定するには、既存の **ScanSetting** オブジェクトを変更します。

```
apiVersion: compliance.openshift.io/v1alpha1
kind: ScanSetting
metadata:
  name: default
  namespace: openshift-compliance
rawResultStorage:
  rotation: 3
  size: 1Gi
roles:
- worker
- master
scanTolerations:
- effect: NoSchedule
  key: node-role.kubernetes.io/master
  operator: Exists
schedule: '0 1 * * *'
timeout: '10m0s' 1
maxRetryOnTimeout: 3 2
```

1. **timeout** 変数は、**1h30m** などの期間文字列として定義されます。デフォルト値は **30m** です。タイムアウトを無効にするには、値を **0s** に設定します。

2. **maxRetryOnTimeout** 変数は、再試行を試行する回数を定義します。デフォルト値は **3** です。

5.6.7.6. サポート

このドキュメントで説明されている手順、または OpenShift Container Platform 全般で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。

カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントについて Jira 課題 を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

5.6.8. oc-compliance プラグインの使用

Compliance Operator はクラスターのチェックおよび修復の多くを自動化しますが、クラスターを準拠させる完全なプロセスでは、管理者が Compliance Operator API や他のコンポーネントと対話する必要があります。**oc-compliance** プラグインはプロセスを容易にします。

5.6.8.1. oc-compliance プラグインのインストール

手順

1. **oc-compliance** イメージを抽出して、**oc-compliance** バイナリーを取得します。

```
$ podman run --rm -v ~/.local/bin:/mnt/out:Z registry.redhat.io/compliance/oc-compliance-rhel8:stable /bin/cp /usr/bin/oc-compliance /mnt/out/
```

出力例

```
W0611 20:35:46.486903   11354 manifest.go:440] Chose linux/amd64 manifest from the manifest list.
```

oc-compliance を実行できるようになりました。

5.6.8.2. 未加工の結果の取得

コンプライアンススキャンが完了すると、個別のチェックの結果は生成される **ComplianceCheckResult** カスタムリソース (CR) にリスト表示されます。ただし、管理者または監査担当者にはスキャンの詳細情報が必要になる場合があります。OpenSCAP ツールは、詳細な結果で Advanced Recording Format (ARF) 形式のファイルを作成します。この ARF ファイルは設定マップまたは他の標準の Kubernetes リソースに保存するには大き過ぎるため、永続ボリューム (PV) がこれを組み込むように作成されます。

手順

- Compliance Operator を使用した PV からの結果の取得は 4 つの手順で実行されます。ただし、**oc-compliance** プラグインでは、単一のコマンドを使用できます。

```
$ oc compliance fetch-raw <object-type> <object-name> -o <output-path>
```

- **<object-type>** は、スキャンの機能に使用されたオブジェクトによって、**scansettingbinding**、**compliancescan** または **compliancesuite** のいずれかにすることができます。
- **<object-name>** は、ARF ファイルを収集に使用するバインディング、スイート、またはスキャンオブジェクトの名前です。**<output-path>** は、結果を配置するローカルディレクトリーです。

以下に例を示します。

```
$ oc compliance fetch-raw scansettingbindings my-binding -o /tmp/
```

出力例

```
Fetching results for my-binding scans: ocp4-cis, ocp4-cis-node-worker, ocp4-cis-node-master
Fetching raw compliance results for scan 'ocp4-cis'.......
The raw compliance results are available in the following directory: /tmp/ocp4-cis
Fetching raw compliance results for scan 'ocp4-cis-node-worker'...........
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-worker
Fetching raw compliance results for scan 'ocp4-cis-node-master'......
The raw compliance results are available in the following directory: /tmp/ocp4-cis-node-master
```

ディレクトリー内のファイルの一覧を表示します。

```
$ ls /tmp/ocp4-cis-node-master/
```

出力例

```
ocp4-cis-node-master-ip-10-0-128-89.ec2.internal-pod.xml.bzip2  ocp4-cis-node-master-ip-10-0-150-5.ec2.internal-pod.xml.bzip2  ocp4-cis-node-master-ip-10-0-163-32.ec2.internal-pod.xml.bzip2
```

結果を抽出します。

```
$ bunzip2 -c resultsdir/worker-scan/worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2 > resultsdir/worker-scan/worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
```

結果を表示します。

```
$ ls resultsdir/worker-scan/
```

出力例

```
worker-scan-ip-10-0-170-231.us-east-2.compute.internal-pod.xml
worker-scan-stage-459-tqkg7-compute-0-pod.xml.bzip2
worker-scan-stage-459-tqkg7-compute-1-pod.xml.bzip2
```

5.6.8.3. スキャンの再実行

スキャンをスケジュールされたジョブとして実行することは可能ですが、多くの場合、修復の適用後、またはクラスターへの他の変更が加えられるなどの場合にとくにスキャンをオンデマンドで再実行する必要があります。

手順

- Compliance Operator でスキャンを再実行するには、スキャンオブジェクトでアノテーションを使用する必要があります。ただし、**oc-compliance** プラグインを使用すると、1 つのコマンドでスキャンを再実行できます。以下のコマンドを入力して、**my-binding** という名前の

ScanSettingBinding オブジェクトのスキャンを再実行します。

```
$ oc compliance rerun-now scansettingbindings my-binding
```

出力例

```
Rerunning scans from 'my-binding': ocp4-cis
Re-running scan 'openshift-compliance/ocp4-cis'
```

5.6.8.4. ScanSettingBinding カスタムリソースの使用

Compliance Operator が提供する **ScanSetting** および **ScanSettingBinding** カスタムリソース (CR) を使用する場合、**schedule**、**machine roles**、**tolerations** などのスキャンオプションのセットを使用している間に、複数のプロファイルに対してスキャンを実行できます。複数の **ComplianceSuite** オブジェクトまたは **ComplianceScan** オブジェクトを使用することがより容易になりますが、新規ユーザーが混乱を生じさせる可能性があります。

oc compliance bind サブコマンドは、**ScanSettingBinding** CR の作成に役立ちます。

手順

1. 以下を実行します。

```
$ oc compliance bind [--dry-run] -N <binding name> [-S <scansetting name>]
<objtype/objname> [..<objtype/objname>]
```

- **-S** フラグを省略する場合、Compliance Operator によって提供される **default** のスキャン設定が使用されます。
- オブジェクトタイプは、Kubernetes オブジェクトタイプです。**profile** または **tailoredprofile** にすることができます。複数のオブジェクトを指定できます。
- オブジェクト名は、**.metadata.name** などの Kubernetes リソースの名前です。
- **--dry-run** オプションを追加して、作成されるオブジェクトの YAML ファイルを表示します。
たとえば、以下のプロファイルとスキャン設定を指定します。

```
$ oc get profile.compliance -n openshift-compliance
```

出力例

```
NAME                 AGE    VERSION
ocp4-cis             3h49m  1.5.0
ocp4-cis-1-4         3h49m  1.4.0
ocp4-cis-1-5         3h49m  1.5.0
ocp4-cis-node        3h49m  1.5.0
ocp4-cis-node-1-4    3h49m  1.4.0
ocp4-cis-node-1-5    3h49m  1.5.0
ocp4-e8              3h49m
ocp4-high            3h49m  Revision 4
ocp4-high-node       3h49m  Revision 4
ocp4-high-node-rev-4 3h49m  Revision 4
```

```
ocp4-high-rev-4            3h49m   Revision 4
ocp4-moderate              3h49m   Revision 4
ocp4-moderate-node         3h49m   Revision 4
ocp4-moderate-node-rev-4   3h49m   Revision 4
ocp4-moderate-rev-4        3h49m   Revision 4
ocp4-nerc-cip              3h49m
ocp4-nerc-cip-node         3h49m
ocp4-pci-dss               3h49m   3.2.1
ocp4-pci-dss-3-2           3h49m   3.2.1
ocp4-pci-dss-4-0           3h49m   4.0.0
ocp4-pci-dss-node          3h49m   3.2.1
ocp4-pci-dss-node-3-2      3h49m   3.2.1
ocp4-pci-dss-node-4-0      3h49m   4.0.0
ocp4-stig                  3h49m   V2R1
ocp4-stig-node             3h49m   V2R1
ocp4-stig-node-v1r1        3h49m   V1R1
ocp4-stig-node-v2r1        3h49m   V2R1
ocp4-stig-v1r1             3h49m   V1R1
ocp4-stig-v2r1             3h49m   V2R1
rhcos4-e8                  3h49m
rhcos4-high                3h49m   Revision 4
rhcos4-high-rev-4          3h49m   Revision 4
rhcos4-moderate            3h49m   Revision 4
rhcos4-moderate-rev-4      3h49m   Revision 4
rhcos4-nerc-cip            3h49m
rhcos4-stig                3h49m   V2R1
rhcos4-stig-v1r1           3h49m   V1R1
rhcos4-stig-v2r1           3h49m   V2R1
```

```
$ oc get scansettings -n openshift-compliance
```

出力例

```
NAME                 AGE
default              10m
default-auto-apply   10m
```

2. **default** 設定を **ocp4-cis** および **ocp4-cis-node** プロファイルに適用するには、以下を実行します。

```
$ oc compliance bind -N my-binding profile/ocp4-cis profile/ocp4-cis-node
```

出力例

```
Creating ScanSettingBinding my-binding
```

ScanSettingBinding CR が作成された後、バインドされたプロファイルは、関連する設定で両方のプロファイルのスキャンを開始します。全体的に、これは Compliance Operator でスキャンを開始するための最も高速な方法です。

5.6.8.5. コントロールの表示

コンプライアンス標準は通常、以下のように階層に編成されます。

- ベンチマークは、特定の標準に関する一連のコントロールの最上位の定義です。例: FedRAMP Moderate または Center for Internet Security (CIS) v.1.6.0。
- コントロールは、ベンチマークへの準拠を確保するために満たさなければならない要件のファミリーを説明します。例: FedRAMP AC-01(アクセス制御ポリシーおよび手順)。
- ルールは、コンプライアンスを取るシステムに固有の単一のチェックでありで、これらの1つ以上のルールがコントロールにマップされます。
- Compliance Operator は、単一のベンチマークに関するプロファイルへのルールのグループ化に対応します。プロファイルの一連のルールの条件を満たすコントロールを判別することが容易ではない場合があります。

手順

- **oc compliance controls** サブコマンドは、指定されたプロファイルが満たす標準およびコントロールのレポートを提供します。

```
$ oc compliance controls profile ocp4-cis-node
```

出力例

```
+-----------+----------+
| FRAMEWORK | CONTROLS |
+-----------+----------+
| CIS-OCP   | 1.1.1    |
+           +----------+
|           | 1.1.10   |
+           +----------+
|           | 1.1.11   |
+           +----------+
...
```

5.6.8.6. コンプライアンス修復の詳細情報の取得

Compliance Operator は、クラスターが準拠できるようにする必要な変更を自動化するために使用される修復オブジェクトを提供します。**fetch-fixes** サブコマンドは、使用する設定の修復を正確に理解するのに役立ちます。**fetch-fixes** サブコマンドを使用して、プロファイル、ルール、または **ComplianceRemediation** オブジェクトから修復オブジェクトを検査対象のディレクトリーに抽出します。

手順

1. プロファイルの修復を表示します。

```
$ oc compliance fetch-fixes profile ocp4-cis -o /tmp
```

出力例

```
No fixes to persist for rule 'ocp4-api-server-api-priority-flowschema-catch-all' 1
No fixes to persist for rule 'ocp4-api-server-api-priority-gate-enabled'
No fixes to persist for rule 'ocp4-api-server-audit-log-maxbackup'
Persisted rule fix to /tmp/ocp4-api-server-audit-log-maxsize.yaml
```

```
No fixes to persist for rule 'ocp4-api-server-audit-log-path'
No fixes to persist for rule 'ocp4-api-server-auth-mode-no-aa'
No fixes to persist for rule 'ocp4-api-server-auth-mode-node'
No fixes to persist for rule 'ocp4-api-server-auth-mode-rbac'
No fixes to persist for rule 'ocp4-api-server-basic-auth'
No fixes to persist for rule 'ocp4-api-server-bind-address'
No fixes to persist for rule 'ocp4-api-server-client-ca'
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-cipher.yaml
Persisted rule fix to /tmp/ocp4-api-server-encryption-provider-config.yaml
```

1 ルールが自動的に修復されないか、修復が行われないために対応する修復のないルールがプロファイルにある場合は常に、**No fixes to persist** の警告が出されることが予想されます。

2. YAML ファイルのサンプルを表示できます。**head** コマンドは、最初の 10 行を表示します。

```
$ head /tmp/ocp4-api-server-audit-log-maxsize.yaml
```

出力例

```yaml
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  maximumFileSizeMegabytes: 100
```

3. スキャン後に作成される **ComplianceRemediation** オブジェクトから修復を表示します。

```
$ oc get complianceremediations -n openshift-compliance
```

出力例

```
NAME                                             STATE
ocp4-cis-api-server-encryption-provider-cipher   NotApplied
ocp4-cis-api-server-encryption-provider-config   NotApplied
```

```
$ oc compliance fetch-fixes complianceremediations ocp4-cis-api-server-encryption-provider-cipher -o /tmp
```

出力例

```
Persisted compliance remediation fix to /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

4. YAML ファイルのサンプルを表示できます。**head** コマンドは、最初の 10 行を表示します。

```
$ head /tmp/ocp4-cis-api-server-encryption-provider-cipher.yaml
```

出力例

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  encryption:
    type: aescbc
```

> **警告**
>
> 修復を直接適用するには注意が必要です。一部の修復は、適度なプロファイルの usbguard ルールなどのように一括して適用できない場合があります。このような場合、Compliance Operator は依存関係に対応し、クラスターは正常な状態のままになるため、ルールを適用できます。

5.6.8.7. ComplianceCheckResult オブジェクトの詳細の表示

スキャンの実行が終了すると、**ComplianceCheckResult** オブジェクトが個別のスキャンルールに作成されます。**view-result** サブコマンドは、**ComplianceCheckResult** オブジェクトの詳細の人間が判読できる出力を提供します。

手順

- 以下を実行します。

```
$ oc compliance view-result ocp4-cis-scheduler-no-bind-address
```

第6章 FILE INTEGRITY OPERATOR

6.1. FILE INTEGRITY OPERATOR の概要

File Integrity Operator は、クラスターノード上でファイル整合性チェックを継続的に実行します。各ノードで特権付きの Advanced Intrusion Detection Environment (AIDE) コンテナーを初期化して実行する DaemonSet をデプロイし、DaemonSet Pod の最初の実行以降に変更されたファイルのログを提供します。

最新の更新は、File Integrity Operator リリースノート を参照してください。

File Integrity Operator のインストール

File Integrity Operator の更新

File Integrity Operator について

カスタム File Integrity Operator の設定

高度なカスタム File Integrity Operator タスクの実行

File Integrity Operator のトラブルシューティング

File Integrity Operator のアンインストール

6.2. FILE INTEGRITY OPERATOR リリースノート

OpenShift Container Platform の File Integrity Operator は、RHCOS ノードでファイル整合性チェックを継続的に実行します。

以下のリリースノートは、OpenShift Container Platform の File Integrity Operator の開発履歴を記録したものです。

File Integrity Operator の概要は、File Integrity Operator について を参照してください。

最新リリースにアクセスするには、File Integrity Operator の更新 を参照してください。

6.2.1. OpenShift File Integrity Operator 1.3.6

OpenShift File Integrity Operator 1.3.6 については、次のアドバイザリーが利用できます。

- RHBA-2025:11535 OpenShift File Integrity Operator Bug Fix Update

6.2.1.1. バグ修正

- 以前は、**oc annotate fileintegrities/<fileintegrity-name> file-integrity.openshift.io/re-init-on-failed=** コマンドを実行すると、すべてのノードで再初期化がトリガーされていました。現在は、障害が発生したノードだけが再初期化されます。(OCPBUGS-18933)

- 以前は、FIO をリセットすると、**NodeHasIntegrityFailure** アラートがクリアされていました。これは、メトリクスの **file_integrity_operator_node_failed** 設定もリセットされたために発生していました。このリリースでは、FIO を再起動しても **NodeHasIntegrityFailure** アラートには影響しません。(OCPBUGS-42807)

- 以前は、**machineset** オブジェクトをスケールアップして新しいノードがクラスターに追加されると、ノードの準備が完了する前に FIO によって新しいノードが **Failed** とマークされていました。このリリースでは、FIO は新しいノードの準備が完了するまで待機します。(**OCPBUGS-36483**)

- 以前は、Advanced Intrusion Detection Environment (AIDE) デーモンセット Pod が、AIDE データベースを常に強制的に初期化していました。このリリースでは、FIO が AIDE データベースを 1 回だけ初期化します。(**OCPBUGS-37300**)

- 以前は、**/hostroot/etc/ipsec.d/openshift.conf** や **hostroot/etc/mco/internal-registry-pull-secret.json** などの Machine Config Operator (MCO) 設定の一部のリンクパスが、MCO の更新中に変更されていました。これにより、更新後にノードに対するファイル整合性チェックが失敗し、ユーザーエクスペリエンスに支障が発生していました。この更新により、MCO 設定内の変更されたファイルリンクパスが更新されました。更新後にファイル整合性チェックに合格するようになり、クラスターの安定性が確保されるようになりました。(**OCPBUGS-41628**)

6.2.2. OpenShift File Integrity Operator 1.3.5

OpenShift File Integrity Operator 1.3.5 については、次のアドバイザリーが利用できます。

- RHBA-2024:10366 OpenShift File Integrity Operator Update

この更新には、基礎となるベースイメージの依存関係のアップグレードが含まれています。

6.2.3. OpenShift File Integrity Operator 1.3.4

OpenShift File Integrity Operator 1.3.4 には、以下のアドバイザリーを利用できます。

- RHBA-2024:2946 OpenShift File Integrity Operator のバグ修正と機能拡張の更新

6.2.3.1. バグ修正

以前は、File Integrity Operator は、複数の証明書のローテーションが原因で **NodeHasIntegrityFailure** アラートを発行していました。このリリースでは、アラートと失敗ステータスが正しくトリガーされるようになりました。(**OCPBUGS-31257**)

6.2.4. OpenShift File Integrity Operator 1.3.3

OpenShift File Integrity Operator 1.3.3 には、以下のアドバイザリーを利用できます。

- RHBA-2023:5652 OpenShift File Integrity Operator のバグ修正と機能拡張の更新

この更新は、基礎となる依存関係の CVE に対処するものです。

6.2.4.1. 新機能および機能拡張

- File Integrity Operator は、FIPS モードで実行されている OpenShift Container Platform クラスターにインストールして使用できます。

重要

クラスターで FIPS モードを有効にするには、FIPS モードで動作するように設定された Red Hat Enterprise Linux (RHEL) コンピューターからインストールプログラムを実行する必要があります。RHEL で FIPS モードを設定する方法の詳細は、RHEL から FIPS モードへの切り替え を参照してください。

FIPS モードでブートされた Red Hat Enterprise Linux (RHEL) または Red Hat Enterprise Linux CoreOS (RHCOS) を実行する場合、OpenShift Container Platform コアコンポーネントは、x86_64、ppc64le、および s390x アーキテクチャーのみで、FIPS 140-2/140-3 検証のために NIST に提出された RHEL 暗号化ライブラリーを使用します。

6.2.4.2. バグ修正

- 以前は、一部の FIO Pod において、プライベートのデフォルトのマウント伝播設定と、**hostPath: path:** / ボリュームマウントを併用すると、マルチパスに依存する CSI ドライバーが壊れるという問題がありました。この問題は修正され、CSI ドライバーは正しく動作するようになりました。(Some OpenShift Operator pods blocking unmounting of CSI volumes when multipath is in use)

- この更新により、CVE-2023-39325 が解決されました。(**CVE-2023-39325**)

6.2.5. OpenShift File Integrity Operator 1.3.2

OpenShift File Integrity Operator 1.3.2 には、以下のアドバイザリーを利用できます。

- RHBA-2023:5107 OpenShift File Integrity Operator Bug Fix Update

この更新は、基礎となる依存関係の CVE に対処するものです。

6.2.6. OpenShift File Integrity Operator 1.3.1

OpenShift File Integrity Operator 1.3.1 には、以下のアドバイザリーを利用できます。

- RHBA-2023:3600 OpenShift File Integrity Operator Bug Fix Update

6.2.6.1. 新機能および機能拡張

- FIO には kubelet 証明書がデフォルトのファイルとして含まれるようになり、OpenShift Container Platform によって管理される場合の警告の発行から除外されます。(**OCPBUGS-14348**)

- FIO が、電子メールを Red Hat テクニカルサポートのアドレスに正しく送信するようになりました。(**OCPBUGS-5023**)

6.2.6.2. バグ修正

- 以前は、ノードがクラスターから削除された場合、FIO は **FileIntegrityNodeStatus** CRD をクリーンアップしませんでした。FIO は、ノードの削除時にノードステータス CRD を正しくクリーンアップするように更新されました。(**OCPBUGS-4321**)

- 以前は、FIO は新しいノードが整合性チェックに失敗したことを誤って示していました。FIO が更新され、クラスターに新しいノードを追加するときにノードステータス CRD が正しく表示されるようになりました。これにより、正しいノードステータス通知が提供されます。

(OCPBUGS-8502)

- 以前は、FIO が **FileIntegrity** CRD を調整しているとき、調整が完了するまでスキャンを一時停止していました。これにより、調整の影響を受けないノード上で過度に積極的な再開始プロセスが発生しました。この問題により、変更される **FileIntegrity** とは関係のないマシン設定プールに不要なデーモンセットも発生します。FIO はこれらのケースを適切に処理し、ファイル整合性の変更の影響を受けるノードの AIDE スキャンのみを一時停止します。(CMP-1097)

6.2.6.3. 既知の問題

FIO 1.3.1 では、IBM Z® クラスター内のノードを増やすと、File Integrity ノードのステータスが **Failed** になる可能性があります。詳細は、Adding nodes in IBM Power® clusters can result in failed File Integrity node status を参照してください。

6.2.7. OpenShift File Integrity Operator 1.2.1

OpenShift File Integrity Operator 1.2.1 には、以下のアドバイザリーを利用できます。

- RHBA-2023:1684 OpenShift File Integrity Operator Bug Fix Update
- このリリースには、更新されたコンテナーの依存関係が含まれています。

6.2.8. OpenShift File Integrity Operator 1.2.0

OpenShift File Integrity Operator 1.2.0 には、以下のアドバイザリーを利用できます。

- RHBA-2023:1273 OpenShift File Integrity Operator 機能拡張の更新

6.2.8.1. 新機能および機能拡張

- File Integrity Operator カスタムリソース (CR) には、最初の AIDE 整合性チェックを開始する前に待機する秒数を指定する **initialDelay** 機能が含まれるようになりました。詳細は、Creating the FileIntegrity custom resource を参照してください。
- File Integrity Operator は安定版になり、リリースチャンネルは **stable** にアップグレードされました。将来のリリースは Semantic Versioning に従います。最新リリースにアクセスするには、File Integrity Operator の更新 を参照してください。

6.2.9. OpenShift File Integrity Operator 1.0.0

OpenShift File Integrity Operator 1.0.0 には、以下のアドバイザリーを利用できます。

- RHBA-2023:0037 OpenShift File Integrity Operator Bug Fix Update

6.2.10. OpenShift File Integrity Operator 0.1.32

OpenShift File Integrity Operator 0.1.32 には、以下のアドバイザリーを利用できます。

- RHBA-2022:7095 OpenShift File Integrity Operator Bug Fix Update

6.2.10.1. バグ修正

- 以前は、File Integrity Operator によって発行されたアラートは namespace を設定していなかったため、アラートが発生した namespace を理解することが困難でした。これで、

Operator は適切な namespace を設定し、アラートに関する詳細情報を提供します。(BZ#2112394)

- 以前は、File Integrity Operator は Operator の起動時にメトリクスサービスを更新しなかったため、メトリクスターゲットに到達できませんでした。今回のリリースでは、File Integrity Operator により、Operator の起動時にメトリクスサービスが確実に更新されるようになりました。(BZ#2115821)

6.2.11. OpenShift File Integrity Operator 0.1.30

OpenShift File Integrity Operator 0.1.30 には、以下のアドバイザリーを利用できます。

- RHBA-2022:5538 OpenShift File Integrity Operator のバグ修正と機能拡張の更新

6.2.11.1. 新機能および機能拡張

- File Integrity Operator は、次のアーキテクチャーでサポートされるようになりました。
  - IBM Power®
  - IBM Z® および IBM® LinuxONE

6.2.11.2. バグ修正

- 以前は、File Integrity Operator が発行したアラートでは namespace が設定されていなかったため、アラートの発生場所を理解することが困難でした。現在、Operator は適切な namespace を設定し、アラートが理解できるように改善されています。(BZ#2101393)

6.2.12. OpenShift File Integrity Operator 0.1.24

OpenShift File Integrity Operator 0.1.24 には、以下のアドバイザリーを利用できます。

- RHBA-2022:1331 OpenShift File Integrity Operator Bug Fix

6.2.12.1. 新機能および機能拡張

- **config.maxBackups** 属性を使用して、**FileIntegrity** カスタムリソース (CR) に保存されるバックアップの最大数を設定できるようになりました。この属性は、ノードに保持するために **re-init** プロセスから残された AIDE データベースおよびログのバックアップの数を指定します。設定された数を超える古いバックアップは自動的にプルーニングされます。デフォルトは 5 つのバックアップに設定されています。

6.2.12.2. バグ修正

- 以前は、Operator を 0.1.21 より古いバージョンから 0.1.22 にアップグレードすると、**re-init** 機能が失敗する可能性がありました。これは、Operator が **configMap** リソースラベルの更新に失敗することが原因でした。現在、最新バージョンにアップグレードすると、リソースラベルが修正されます。(BZ#2049206)
- 以前は、デフォルトの **configMap** スクリプトの内容を適用するときに、間違ったデータキーが比較されていました。これにより、Operator のアップグレード後に **aide-reinit** スクリプトが適切に更新されず、**re-init** プロセスが失敗することがありました。これで、**daemonSets** が完了するまで実行され、AIDE データベースの **re-init** プロセスが正常に実行されます。(BZ#2072058)

6.2.13. OpenShift File Integrity Operator 0.1.22

OpenShift File Integrity Operator 0.1.22 には、以下のアドバイザリーを利用できます。

- RHBA-2022:0142 OpenShift File Integrity Operator のバグ修正

6.2.13.1. バグ修正

- 以前は、File Integrity Operator がインストールされているシステムでは、**/etc/kubernetes/aide.reinit** ファイルが原因で、OpenShift Container Platform の更新が中断される可能性がありました。これは、**/etc/kubernetes/aide.reinit** ファイルが存在したが、後で **ostree** 検証の前に削除された場合に発生しました。今回の更新では、**/etc/kubernetes/aide.reinit** が **/run** ディレクトリーに移動し、OpenShift Container Platform の更新と競合しないようになっています。(BZ#2033311)

6.2.14. OpenShift File Integrity Operator 0.1.21

OpenShift File Integrity Operator 0.1.21 には、以下のアドバイザリーを利用できます。

- RHBA-2021:4631 OpenShift File Integrity Operator のバグ修正と機能拡張の更新

6.2.14.1. 新機能および機能拡張

- **FileIntegrity** スキャン結果および処理メトリックに関連するメトリックは、Web コンソールの監視ダッシュボードに表示されます。結果には、**file_integrity_operator_** の接頭辞が付けられます。
- ノードの整合性障害が 1 秒を超えると、Operator の namespace で提供されるデフォルトの **PrometheusRule** が警告を発します。
- 次の動的な Machine Config Operator および Cluster Version Operator 関連のファイルパスは、ノードの更新中の誤検知を防ぐために、デフォルトの AIDE ポリシーから除外されています。
  - /etc/machine-config-daemon/currentconfig
  - /etc/pki/ca-trust/extracted/java/cacerts
  - /etc/cvo/updatepayloads
  - /root/.kube
- AIDE デーモンプロセスは v0.1.16 よりも安定性が向上しており、AIDE データベースの初期化時に発生する可能性のあるエラーに対する耐性が高くなっています。

6.2.14.2. バグ修正

- 以前は、Operator が自動的にアップグレードしたときに、古いデーモンセットは削除されませんでした。このリリースでは、自動アップグレード中に古いデーモンセットが削除されます。

6.2.15. 関連情報

- File Integrity Operator について

6.3. FILE INTEGRITY OPERATOR のサポート

6.3.1. File Integrity Operator のライフサイクル

File Integrity Operator は "Rolling Stream" Operator で、OpenShift Container Platform リリースの更新が非同期で利用可能であることを意味します。詳細は、Red Hat カスタマーポータルの OpenShift Operator ライフサイクル を参照してください。

6.3.2. サポート

このドキュメントで説明されている手順、または OpenShift Container Platform 全般で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。

カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントについて Jira 課題 を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

6.4. FILE INTEGRITY OPERATOR のインストール

重要

この Operator を適切に機能させるには、すべてのクラスターノードのリリースバージョンが同じである必要があります。たとえば、RHCOS を実行しているノードの場合、すべてのノードの RHCOS バージョンが同じである必要があります。

6.4.1. Web コンソールでの File Integrity Operator のインストール

前提条件

- **admin** 権限がある。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. File Integrity Operator を検索し、**Install** をクリックします。
3. **Installation mode** および **namespace** のデフォルトの選択を維持し、Operator が **openshift-file-integrity** namespace にインストールされていることを確認します。
4. **Install** をクリックします。

検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators → Installed Operators** ページに移動します。
2. Operator が **openshift-file-integrity** namespace にインストールされており、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators → Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。
2. **Workloads → Pods** ページに移動し、**openshift-file-integrity** プロジェクトの Pod で問題を報告しているログの有無を確認します。

6.4.2. CLI を使用した File Integrity Operator のインストール

前提条件

- **admin** 権限がある。

手順

1. 以下を実行して **Namespace** オブジェクト YAML ファイルを作成します。

```
$ oc create -f <file-name>.yaml
```

出力例

```
apiVersion: v1
kind: Namespace
metadata:
  labels:
    openshift.io/cluster-monitoring: "true"
    pod-security.kubernetes.io/enforce: privileged 1
  name: openshift-file-integrity
```

1 OpenShift Container Platform 4.16 では、Pod セキュリティーラベルを namespace レベルで **privileged** に設定する必要があります。

2. **OperatorGroup** オブジェクト YAML ファイルを作成します。

```
$ oc create -f <file-name>.yaml
```

出力例

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
```

```
spec:
  targetNamespaces:
  - openshift-file-integrity
```

3. **Subscription** オブジェクト YAML ファイルを作成します。

```
$ oc create -f <file-name>.yaml
```

出力例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: file-integrity-operator
  namespace: openshift-file-integrity
spec:
  channel: "stable"
  installPlanApproval: Automatic
  name: file-integrity-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

検証

1. CSV ファイルを確認して、インストールが正常に完了したことを確認します。

```
$ oc get csv -n openshift-file-integrity
```

2. File Integrity Operator が稼働していることを確認します。

```
$ oc get deploy -n openshift-file-integrity
```

6.4.3. 関連情報

- File Integrity Operator はネットワークが制限された環境でサポートされています。詳細は、ネットワークが制限された環境での Operator Lifecycle Manager の使用 を参照してください。

6.5. FILE INTEGRITY OPERATOR の更新

クラスター管理者は、OpenShift Container Platform クラスターで File Integrity Operator を更新できます。

6.5.1. Operator 更新の準備

インストールされた Operator のサブスクリプションは、Operator の更新を追跡および受信する更新チャネルを指定します。更新チャネルを変更して、新しいチャネルからの更新の追跡と受信を開始できます。

サブスクリプションの更新チャネルの名前は Operators 間で異なる可能性がありますが、命名スキーム通常、特定の Operators 内の共通の規則に従います。たとえば、チャネル名は Operator によって提供されるアプリケーションのマイナーリリース更新ストリーム (**1.2**、**1.3**) またはリリース頻度

(**stable**、**fast**) に基づく可能性があります。

注記

インストールされた Operators は、現在のチャネルよりも古いチャネルに切り換えることはできません。

Red Hat Customer Portal Labs には、管理者が Operators の更新を準備するのに役立つ以下のアプリケーションが含まれています。

- Red Hat OpenShift Container Platform Operator Update Information Checker

このアプリケーションを使用して、Operator Lifecycle Manager ベースの Operator を検索し、OpenShift Container Platform の異なるバージョン間で更新チャネルごとに利用可能な Operator バージョンを確認できます。Cluster Version Operator ベースの Operator は含まれません。

6.5.2. Operator の更新チャネルの変更

OpenShift Container Platform Web コンソールを使用して、Operator の更新チャネルを変更できます。

ヒント

サブスクリプションの承認ストラテジーが **Automatic** に設定されている場合、アップグレードプロセスは、選択したチャネルで新規 Operator バージョンが利用可能になるとすぐに開始します。承認ストラテジーが **Manual** に設定されている場合は、保留中のアップグレードを手動で承認する必要があります。

前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

手順

1. Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. 更新チャネルを変更する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。
4. **Update channel** の下にある更新チャネルの名前をクリックします。
5. 変更する新しい更新チャネルをクリックし、**Save** をクリックします。
6. **Automatic** 承認ストラテジーのあるサブスクリプションの場合、更新は自動的に開始します。**Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。
   Manual 承認ストラテジーのあるサブスクリプションの場合、**Subscription** タブから更新を手動で承認できます。

6.5.3. 保留中の Operator 更新の手動による承認

インストールされた Operator のサブスクリプションの承認ストラテジーが **Manual** に設定されている場合、新規の更新が現在の更新チャネルにリリースされると、インストールを開始する前に更新を手動で承認する必要があります。

前提条件

- Operator Lifecycle Manager (OLM) を使用して以前にインストールされている Operator。

手順

1. OpenShift Container Platform Web コンソールの **Administrator** パースペクティブで、**Operators → Installed Operators** に移動します。
2. 更新が保留中の Operator は **Upgrade available** のステータスを表示します。更新する Operator の名前をクリックします。
3. **Subscription** タブをクリックします。承認が必要な更新は、**Upgrade status** の横に表示されます。たとえば、**1 requires approval** が表示される可能性があります。
4. **1 requires approval** をクリックしてから、**Preview Install Plan** をクリックします。
5. 更新に利用可能なリソースとして一覧表示されているリソースを確認します。問題がなければ、**Approve** をクリックします。
6. **Operators → Installed Operators** ページに戻り、更新の進捗をモニターします。完了時に、ステータスは **Succeeded** および **Up to date** に変更されます。

6.6. FILE INTEGRITY OPERATOR について

File Integrity Operator は OpenShift Container Platform Operator であり、クラスターノード上でファイルの整合性チェックを継続的に実行します。これは、各ノードで特権付きの AIDE (advanced intrusion detection environment; 高度な侵入検知環境) コンテナーを各ノードで初期化し、実行するデーモンセットをデプロイし、ステータスオブジェクトをデーモンセット Pod の初回実行時に変更されるファイルのログと共に提供します。

重要

現時点では、Red Hat Enterprise Linux CoreOS (RHCOS) ノードのみがサポートされます。

6.6.1. FileIntegrity カスタムリソースの作成

FileIntegrity カスタムリソース (CR) のインスタンスは、1 つ以上のノードの継続的なファイル整合性スキャンのセットを表します。

それぞれの **FileIntegrity** CR は、**FileIntegrity** CR 仕様に一致するノード上で AIDE を実行するデーモンセットによってサポートされます。

手順

1. **worker-fileintegrity.yaml** という名前の次の例の **FileIntegrity** CR を作成して、ワーカーノードでのスキャンを有効にします。

   サンプル FileIntegrity CR

   ```
   -
   ```

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: worker-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector: 1
      node-role.kubernetes.io/worker: ""
  tolerations: 2
  - key: "myNode"
    operator: "Exists"
    effect: "NoSchedule"
  config: 3
    name: "myconfig"
    namespace: "openshift-file-integrity"
    key: "config"
    gracePeriod: 20 4
    maxBackups: 5 5
    initialDelay: 60 6
  debug: false
status:
  phase: Active 7
```

1. ノードスキャンをスケジュールするためのセレクターを定義します。
2. カスタムテイントを持つノードにスケジュールする **tolerations** を指定します。指定しない場合、メインノードとインフラノードでの実行を許可するデフォルトの容認が適用されます。
3. 使用する AIDE 設定を含む **ConfigMap** を定義します。
4. AIDE 整合性チェックの間に一時停止する秒数。ノード上で AIDE チェックを頻繁に実行すると、多くのリソースが消費する可能性があるため、間隔をより長く指定することができます。デフォルトは 900 秒 (15 分) です。
5. ノードで保持する re-init プロセスから残った AIDE データベースとログのバックアップの最大数。この数を超える古いバックアップは、デーモンによって自動的に削除されます。デフォルトは 5 に設定されています。
6. 最初の AIDE 整合性チェックを開始するまで待機する秒数。デフォルトは 0 に設定されています。
7. **FileIntegrity** インスタンスの実行ステータス。ステータスは、**Initializing**、**Pending**、または **Active** です。

Initializing	**FileIntegrity** オブジェクトは現在、AIDE データベースを初期化または再初期化しています。
Pending	**FileIntegrity** デプロイメントはまだ作成中です。
Active	スキャンはアクティブで進行中です。

2. YAML ファイルを **openshift-file-integrity** namespace に適用します。

```
$ oc apply -f worker-fileintegrity.yaml -n openshift-file-integrity
```

検証

- 次のコマンドを実行して、**FileIntegrity** オブジェクトが正常に作成されたことを確認します。

```
$ oc get fileintegrities -n openshift-file-integrity
```

出力例

```
NAME                  AGE
worker-fileintegrity   14s
```

6.6.2. FileIntegrity カスタムリソースのステータスの確認

FileIntegrity カスタムリソース (CR) は、**.status.phase** サブリソースでそのステータスを報告します。

手順

- **FileIntegrity** CR ステータスをクエリーするには、以下を実行します。

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status.phase }"
```

出力例

```
Active
```

6.6.3. FileIntegrity カスタムリソースの各種フェーズ

- **Pending**: カスタムリソース (CR) の作成後のフェーズ。
- **Active**: バッキングデーモンセットが実行するフェーズ。
- **Initializing**: AIDE データベースが再初期化されるフェーズ。

6.6.4. FileIntegrityNodeStatuses オブジェクトについて

FileIntegrity CR のスキャン結果は、**FileIntegrityNodeStatuses** という別のオブジェクトで報告されます。

```
$ oc get fileintegritynodestatuses
```

出力例

```
NAME                                                AGE
worker-fileintegrity-ip-10-0-130-192.ec2.internal   101s
worker-fileintegrity-ip-10-0-147-133.ec2.internal   109s
worker-fileintegrity-ip-10-0-165-160.ec2.internal   102s
```

> **注記**
>
> **FileIntegrityNodeStatus** オブジェクトの結果が利用可能になるまで、しばらく時間がかかる場合があります。

ノードごとに 1 つの結果オブジェクトがあります。それぞれの **FileIntegrityNodeStatus** オブジェクトの **nodeName** 属性は、スキャンされるノードに対応します。ファイル整合性スキャンのステータスは、スキャン条件を保持する **results** 配列で表示されます。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

fileintegritynodestatus オブジェクトは AIDE 実行の最新のステータスを報告し、**status** フィールドに **Failed**、**Succeeded**、または **Errored** などのステータスを公開します。

```
$ oc get fileintegritynodestatuses -w
```

出力例

```
NAME                                                    NODE                                          STATUS
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-169-137.us-east-2.compute.internal   ip-10-0-169-137.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-
2.compute.internal    Failed
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-134-186.us-east-2.compute.internal   ip-10-0-134-186.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-222-188.us-east-2.compute.internal   ip-10-0-222-188.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-194-66.us-east-2.compute.internal    ip-10-0-194-66.us-east-
2.compute.internal    Failed
example-fileintegrity-ip-10-0-150-230.us-east-2.compute.internal   ip-10-0-150-230.us-east-
2.compute.internal   Succeeded
example-fileintegrity-ip-10-0-180-200.us-east-2.compute.internal   ip-10-0-180-200.us-east-
2.compute.internal   Succeeded
```

6.6.5. FileIntegrityNodeStatus CR ステータスタイプ

これらの条件は、対応する **FileIntegrityNodeStatus** CR ステータスの results 配列で報告されます。

- **Succeeded**: 整合性チェックにパスしました。データベースが最後に初期化されてから、AIDE チェックの対象となるファイルおよびディレクトリーは変更されていません。
- **Failed**: 整合性チェックにパスしません。データベースが最後に初期化されてから、AIDE チェックの対象となるファイルまたはディレクトリーが変更されています。
- **Errored**: AIDE スキャナーで内部エラーが発生しました。

6.6.5.1. FileIntegrityNodeStatus CR の成功例

成功ステータスのある状態の出力例

```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:57Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:46:03Z"
  }
]
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:45:48Z"
  }
]
```

この場合、3 つのスキャンがすべて正常に実行され、現時点ではその他の状態は存在しません。

6.6.5.2. FileIntegrityNodeStatus CR の失敗ステータスの例

障害のある状態をシミュレートするには、AIDE が追跡するファイルの 1 つを変更します。たとえば、ワーカーノードのいずれかで **/etc/resolv.conf** を変更します。

```
$ oc debug node/ip-10-0-130-192.ec2.internal
```

出力例

```
Creating debug namespace/openshift-debug-node-ldfbj ...
Starting pod/ip-10-0-130-192ec2internal-debug ...
To use host binaries, run `chroot /host`
Pod IP: 10.0.130.192
If you don't see a command prompt, try pressing enter.
sh-4.2# echo "# integrity test" >> /host/etc/resolv.conf
sh-4.2# exit

Removing debug pod ...
Removing debug namespace/openshift-debug-node-ldfbj ...
```

しばらくすると、**Failed** 状態が対応する **FileIntegrityNodeStatus** オブジェクトの results 配列で報告されます。前回の **Succeeded** 状態は保持されるため、チェックが失敗した時間を特定することができます。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io/worker-fileintegrity-ip-10-0-130-
192.ec2.internal -ojsonpath='{.results}' | jq -r
```

または、オブジェクト名に言及していない場合は、以下を実行します。

```
$ oc get fileintegritynodestatuses.fileintegrity.openshift.io -ojsonpath='{.items[*].results}' | jq
```

出力例


```
[
  {
    "condition": "Succeeded",
    "lastProbeTime": "2020-09-15T12:54:14Z"
  },
  {
    "condition": "Failed",
    "filesChanged": 1,
    "lastProbeTime": "2020-09-15T12:57:20Z",
    "resultConfigMapName": "aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed",
    "resultConfigMapNamespace": "openshift-file-integrity"
  }
]
```


Failed 状態は、失敗した内容と理由に関する詳細を示す設定マップを参照します。

```
$ oc describe cm aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
```

出力例

```
Name:         aide-ds-worker-fileintegrity-ip-10-0-130-192.ec2.internal-failed
Namespace:    openshift-file-integrity
Labels:       file-integrity.openshift.io/node=ip-10-0-130-192.ec2.internal
              file-integrity.openshift.io/owner=worker-fileintegrity
              file-integrity.openshift.io/result-log=
Annotations:  file-integrity.openshift.io/files-added: 0
              file-integrity.openshift.io/files-changed: 1
              file-integrity.openshift.io/files-removed: 0

Data

integritylog:
------
AIDE 0.15.1 found differences between database and filesystem!!
Start timestamp: 2020-09-15 12:58:15

Summary:
  Total number of files:  31553
  Added files:            0
  Removed files:          0
  Changed files:          1


---------------------------------------------------
Changed files:
---------------------------------------------------

changed: /hostroot/etc/resolv.conf

---------------------------------------------------
```

```
Detailed information about changes:
---------------------------------------------------



File: /hostroot/etc/resolv.conf
 SHA512   : sTQYpB/AL7FeoGtu/1g7opv6C+KT1CBJ , qAeM+a8yTgHPnIHMaRlS+so61EN8VOpg

Events:  <none>
```

設定マップのデータサイズの制限により、1 MB を超える AIDE ログが base64 でエンコードされた gzip アーカイブとして障害用の設定マップに追加されます。ログを抽出するには、次のコマンドを使用します。

```
$ oc get cm <failure-cm-name> -o json | jq -r '.data.integritylog' | base64 -d | gunzip
```

> **注記**
>
> 圧縮されたログの有無は、設定マップにある **file-integrity.openshift.io/compressed** アノテーションキーで示唆されます。

6.6.6. イベントについて

FileIntegrity および **FileIntegrityNodeStatus** オブジェクトのステータスの移行は **イベント** でログに記録されます。イベントの作成時間は、**Initializing** から **Active** など、最新の移行を反映し、必ずしも最新のスキャン結果ではありません。ただし、最新のイベントは常に最新のステータスを反映しています。

```
$ oc get events --field-selector reason=FileIntegrityStatus
```

出力例

```
LAST SEEN   TYPE     REASON              OBJECT                            MESSAGE
97s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Pending
67s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Initializing
37s         Normal   FileIntegrityStatus   fileintegrity/example-fileintegrity   Active
```

ノードのスキャンに失敗すると、イベントは **add/changed/removed** および設定マップ情報と共に作成されます。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

出力例

```
LAST SEEN   TYPE      REASON              OBJECT                            MESSAGE
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-
0-134-173.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-
0-168-238.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-
0-169-175.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-
0-152-92.ec2.internal
```

```
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-158-144.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-131-30.ec2.internal
87m         Warning   NodeIntegrityStatus   fileintegrity/example-fileintegrity   node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

ノードのステータスが移行されなかった場合でも、追加、変更、または削除されたファイルの数を変更すると、新しいイベントが生成されます。

```
$ oc get events --field-selector reason=NodeIntegrityStatus
```

出力例

```
LAST SEEN   TYPE      REASON                OBJECT                                MESSAGE
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-134-173.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-168-238.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-169-175.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-152-92.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-158-144.ec2.internal
114m        Normal    NodeIntegrityStatus   fileintegrity/example-fileintegrity   no changes to node ip-10-0-131-30.ec2.internal
87m         Warning   NodeIntegrityStatus   fileintegrity/example-fileintegrity   node ip-10-0-152-92.ec2.internal has changed! a:1,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
40m         Warning   NodeIntegrityStatus   fileintegrity/example-fileintegrity   node ip-10-0-152-92.ec2.internal has changed! a:3,c:1,r:0 \ log:openshift-file-integrity/aide-ds-example-fileintegrity-ip-10-0-152-92.ec2.internal-failed
```

6.7. カスタム FILE INTEGRITY OPERATOR の設定

6.7.1. FileIntegrity オブジェクト属性の表示

Kubernetes カスタムリソース (CR) の場合と同様に、**oc explain fileintegrity** を実行してから、以下を使用して個別の属性を参照できます。

```
$ oc explain fileintegrity.spec
```

```
$ oc explain fileintegrity.spec.config
```

6.7.2. 重要な属性

表6.1 重要な spec および spec.config 属性

属性	説明
spec.nodeSelector	AIDE Pod が該当ノードでスケジュール対象にするために使用する、ノードのラベルと一致する必要があるキーと値のペアのマップ。通常は、**node-role.kubernetes.io/worker: ""** がすべてのワーカーノードで AIDE をスケジュールし、**node.openshift.io/os_id: "rhcos"** がすべての Red Hat Enterprise Linux CoreOS (RHCOS) ノードでスケジュールする単一のキーと値のペアのみを設定します。
spec.debug	ブール値の属性。**true** に設定すると、AIDE デーモンセットの Pod で実行されるデーモンは追加の情報を出力します。
spec.tolerations	カスタムテイントを持つノードにスケジュールする容認を指定します。指定されない場合は、デフォルトの toleration が適用され、これにより容認はコントロールプレーンノードで実行できます。
spec.config.gracePeriod	AIDE 整合性チェックの間に一時停止する秒数。ノード上で AIDE チェックを頻繁に実行すると、多くのリソースが消費する可能性があるため、間隔をより長く指定することができます。デフォルトは **900** (15 分) です。
maxBackups	ノードで保持する **re-init** 化プロセスから残った AIDE データベースとログのバックアップの最大数。この数を超える古いバックアップは、デーモンによって自動的に削除されます。
spec.config.name	カスタム AIDE 設定を含む configMap の名前。省略した場合、デフォルトの設定が作成されます。
spec.config.namespace	カスタム AIDE 設定を含む configMap の namespace。設定されていない場合、FIO は RHCOS システムに適したデフォルト設定を生成します。
spec.config.key	**name** と **namespace** で指定された設定マップに実際の AIDE 設定を含むキー。デフォルト値は **aide.conf** です。
spec.config.initialDelay	最初の AIDE 整合性チェックを開始するまで待機する秒数。デフォルトは 0 に設定されています。この属性は任意です。

6.7.3. デフォルト設定の確認

デフォルトの File Integrity Operator 設定は、**FileIntegrity** CR と同じ名前で設定マップに保存されます。

手順

- デフォルトの設定を確認するには、以下を実行します。

```
$ oc describe cm/worker-fileintegrity
```

6.7.4. デフォルトの File Integrity Operator 設定について

以下は、設定マップの **aide.conf** キーの抜粋です。

```
@@define DBDIR /hostroot/etc/kubernetes
@@define LOGDIR /hostroot/etc/kubernetes
database=file:@@{DBDIR}/aide.db.gz
database_out=file:@@{DBDIR}/aide.db.gz
gzip_dbout=yes
verbose=5
report_url=file:@@{LOGDIR}/aide.log
report_url=stdout
PERMS = p+u+g+acl+selinux+xattrs
CONTENT_EX = sha512+ftype+p+u+g+n+acl+selinux+xattrs

/hostroot/boot/     CONTENT_EX
/hostroot/root/\..* PERMS
/hostroot/root/   CONTENT_EX
```

FileIntegrity インスタンスのデフォルト設定は、以下のディレクトリー下にあるファイルの範囲を指定します。

- **/root**
- **/boot**
- **/usr**
- **/etc**

以下のディレクトリーは対象外です。

- **/var**
- **/opt**
- **/etc/** 内の OpenShift Container Platform 固有の除外対象

6.7.5. カスタム AIDE 設定の指定

DBDIR、**LOGDIR**、**database**、および **database_out** などの AIDE 内部動作を設定するエントリーは Operator によって上書きされます。Operator は、整合性に関する変更の有無についてすべてのパスを監視できるよう接頭辞を **/hostroot/** に追加します。これにより、コンテナー化された環境用にカスタマイズされない既存の AIDE 設定を再使用や、ルートディレクトリーからの開始がより容易になります。

> **注記**
>
> **/hostroot** は、AIDE を実行する Pod がホストのファイルシステムをマウントするディレクトリーです。設定を変更すると、データベースの再初期化がトリガーされます。

6.7.6. カスタム File Integrity Operator 設定の定義

この例では、**worker-fileintegrity** CR に提供されるデフォルト設定に基づいてコントロールプレーンノードで実行されるスキャナーのカスタム設定を定義することに重点を置いています。このワークフローは、デーモンセットとして実行されているカスタムソフトウェアをデプロイし、そのデータをコントロールプレーンノードの **/opt/mydaemon** の下に保存する場合に役立ちます。

手順

1. デフォルト設定のコピーを作成します。
2. デフォルト設定を、監視するか、除外する必要があるファイルで編集します。
3. 編集したコンテンツを新たな設定マップに保存します。
4. **spec.config** の属性を使用して、**FileIntegrity** オブジェクトを新規の設定マップにポイントします。
5. デフォルト設定を抽出します。

   ```
   $ oc extract cm/worker-fileintegrity --keys=aide.conf
   ```

   これにより、編集可能な **aide.conf** という名前のファイルが作成されます。この例では、Operator のパスの後処理方法を説明するために、接頭辞なしで除外ディレクトリーを追加します。

   ```
   $ vim aide.conf
   ```

   出力例

   ```
   /hostroot/etc/kubernetes/static-pod-resources
   !/hostroot/etc/kubernetes/aide.*
   !/hostroot/etc/kubernetes/manifests
   !/hostroot/etc/docker/certs.d
   !/hostroot/etc/selinux/targeted
   !/hostroot/etc/openvswitch/conf.db
   ```

   コントロールプレーンノードに固有のパスを除外します。

   ```
   !/opt/mydaemon/
   ```

   その他のコンテンツを **/etc** に保存します。

   ```
   /hostroot/etc/ CONTENT_EX
   ```

6. このファイルに基づいて設定マップを作成します。

   ```
   $ oc create cm master-aide-conf --from-file=aide.conf
   ```

7. 設定マップを参照する **FileIntegrity** CR マニフェストを定義します。

```
apiVersion: fileintegrity.openshift.io/v1alpha1
kind: FileIntegrity
metadata:
  name: master-fileintegrity
  namespace: openshift-file-integrity
spec:
  nodeSelector:
      node-role.kubernetes.io/master: ""
  config:
      name: master-aide-conf
      namespace: openshift-file-integrity
```

Operator は指定された設定マップファイルを処理し、結果を **FileIntegrity** オブジェクトと同じ名前の設定マップに保存します。

```
$ oc describe cm/master-fileintegrity | grep /opt/mydaemon
```

出力例

```
!/hostroot/opt/mydaemon
```

6.7.7. カスタムのファイル整合性設定の変更

ファイル整合性の設定を変更するには、生成される設定マップを変更しないでください。その代わりに、**spec.name**、**namespace**、および **key** 属性を使用して **FileIntegrity** オブジェクトにリンクされる設定マップを変更します。

6.8. 高度なカスタム FILE INTEGRITY OPERATOR タスクの実行

6.8.1. データベースの再初期化

File Integrity Operator が予定される変更を検知する場合、データベースの再初期化が必要になることがあります。

手順

- **FileIntegrity** カスタムリソース (CR) に **file-integrity.openshift.io/re-init** のアノテーションを付けます。

```
$ oc annotate fileintegrities/worker-fileintegrity file-integrity.openshift.io/re-init=
```

古いデータベースとログファイルがバックアップされ、新しいデータベースが初期化されます。**oc debug** を使用して起動する Pod の以下の出力に示されるように、古いデータベースおよびログは **/etc/kubernetes** の下にあるノードに保持されます。

出力例

```
 ls -lR /host/etc/kubernetes/aide.*
-rw-------. 1 root root 1839782 Sep 17 15:08 /host/etc/kubernetes/aide.db.gz
-rw-------. 1 root root 1839783 Sep 17 14:30 /host/etc/kubernetes/aide.db.gz.backup-
```

```
20200917T15_07_38
-rw-------. 1 root root   73728 Sep 17 15:07 /host/etc/kubernetes/aide.db.gz.backup-
20200917T15_07_55
-rw-r--r--. 1 root root       0 Sep 17 15:08 /host/etc/kubernetes/aide.log
-rw-------. 1 root root     613 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_38
-rw-r--r--. 1 root root       0 Sep 17 15:07 /host/etc/kubernetes/aide.log.backup-
20200917T15_07_55
```

レコードの永続性を確保するために、生成される config map は **FileIntegrity** オブジェクトによって所有されません。そのため、手動のクリーンアップが必要になります。結果として、以前の整合性の失敗が依然として **FileIntegrityNodeStatus** オブジェクトに表示されます。

6.8.2. マシン設定の統合

OpenShift Container Platform 4 では、クラスターノード設定は **MachineConfig** オブジェクトで提供されます。**MachineConfig** オブジェクトによって生じるファイルへの変更が予想されますが、ファイルの整合性スキャンは失敗しないことを前提にすることができます。**MachineConfig** オブジェクトの更新によって生じるファイルの変更を抑制するために、File Integrity Operator はノードオブジェクトを監視します。ノードが更新されると、AIDE スキャンは更新の期間一時停止します。更新が完了すると、データベースが再初期化され、スキャンが再開されます。

この一時停止および再開ロジックは、ノードオブジェクトのアノテーションに反映されるため、**MachineConfig** API での更新にのみ適用されます。

6.8.3. デーモンセットの参照

それぞれの **FileIntegrity** オブジェクトはノード数に関するスキャンを表します。スキャン自体は、デーモンセットによって管理される Pod で実行されます。

FileIntegrity オブジェクトを表すデーモンセットを見つけるには、以下を実行します。

```
$ oc -n openshift-file-integrity get ds/aide-worker-fileintegrity
```

そのデーモンセットの Pod をリスト表示するには、以下を実行します。

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

単一の AIDE Pod のログを表示するには、Pod のいずれかで **oc logs** を呼び出します。

```
$ oc -n openshift-file-integrity logs pod/aide-worker-fileintegrity-mr8x6
```

出力例

```
Starting the AIDE runner daemon
initializing AIDE db
initialization finished
running aide check
...
```

AIDE デーモンによって作成された config map は保持されず、File Integrity Operator がこれらを処理した後に削除されます。ただし、障害およびエラーの発生時に、これらの config map の内容は **FileIntegrityNodeStatus** オブジェクトが参照する設定マップにコピーされます。

6.9. FILE INTEGRITY OPERATOR のトラブルシューティング

6.9.1. 一般的なトラブルシューティング

問題

File Integrity Operator の問題をトラブルシューティングする必要がある場合があります。

解決策

FileIntegrity オブジェクトでデフォルトフラグを有効にします。**debug** フラグは、**DaemonSet** Pod で実行されるデーモンの詳細度を上げ、AIDE チェックを実行します。

6.9.2. AIDE 設定の確認

問題

AIDE 設定を確認する必要がある場合があります。

解決策

AIDE 設定は、**FileIntegrity** オブジェクトと同じ名前で設定マップに保存されます。すべての AIDE 設定の config map には、**file-integrity.openshift.io/aide-conf** のラベルが付けられます。

6.9.3. FileIntegrity オブジェクトのフェーズの判別

問題

FileIntegrity オブジェクトが存在するかどうかを判別し、その現在のステータスを確認する必要がある場合があります。

解決策

FileIntegrity オブジェクトの現在のステータスを確認するには、以下を実行します。

```
$ oc get fileintegrities/worker-fileintegrity  -o jsonpath="{ .status }"
```

FileIntegrity オブジェクトおよびサポートするデーモンセットが作成されると、ステータスは **Active** に切り替わります。切り替わらない場合は、Operator Pod ログを確認してください。

6.9.4. デーモンセットの Pod が予想されるノードで実行されていることの判別

問題

デーモンセットが存在し、その Pod が実行されることが予想されるノードで実行されていることを確認する必要がある場合があります。

解決策

以下を実行します。

```
$ oc -n openshift-file-integrity get pods -lapp=aide-worker-fileintegrity
```

注記

-owide を追加すると、Pod が実行されているノードの IP アドレスが含まれます。

デーモン Pod のログを確認するには、**oc logs** を実行します。

AIDE コマンドの戻り値をチェックして、チェックが合否を確認します。

6.10. FILE INTEGRITY OPERATOR のアンインストール

OpenShift Container Platform Web コンソールを使用して、クラスターから File Integrity Operator を削除できます。

6.10.1. Web コンソールを使用した File Integrity Operator のアンインストール

File Integrity Operator を削除するには、まずすべての namespace 内の **FileIntegrity** オブジェクトを削除する必要があります。オブジェクトを削除したら、Operator とその namespace を削除できます。

前提条件

- **cluster-admin** 権限を持つアカウントを使用する OpenShift Container Platform クラスターにアクセスできる。
- File Integrity Operator がインストールされています。

手順

1. **Operators** → **Installed Operators** → **File Integrity Operator** ページに移動します。
2. **File Integrity** タブで、**Show operands in: All namespaces**デフォルトオプションが選択され、すべての namespace 内のすべての **FileIntegrity** オブジェクトがリスト表示されていることを確認します。
3. Options メニュー をクリックしてから、**Delete FileIntegrity** をクリックして、**FileIntegrity** オブジェクトを削除します。すべての **FileIntegrity** オブジェクトが削除されていることを確認します。
4. **Administration** → **Operators** → **Installed Operators** ページに移動します。
5. **File Integrity Operator** エントリーの Options メニュー をクリックして **Uninstall Operator** を選択します。
6. **Home** → **Projects** ページに移動します。
7. **openshift-file-integrity** を検索します。
8. **openshift-file-integrity** プロジェクトエントリーの Options メニュー をクリックし、**Delete Project** をクリックします。Web コンソールで **Delete Project** ダイアログボックスが開きます。

検証

- **Delete Project** ダイアログボックスに **openshift-file-integrity** と入力し、**Delete** ボタンをクリックします。

第7章 SECURITY PROFILES OPERATOR

7.1. セキュリティープロファイルオペレータの概要

OpenShift Container Platform Security Profiles Operator (SPO) は、セキュアコンピューティング (seccomp) プロファイルおよび SELinux プロファイルをカスタムリソースとして定義し、プロファイルを特定の namespace のすべてのノードに同期する方法を提供します。最新の更新は、リリースノート を参照してください。

SPO はカスタムリソースを各ノードに配布できますが、調整ループによってプロファイルが最新の状態に保たれます。Security Profiles Operator について を参照してください。

SPO は、namespace に属するワークロードの SELinux ポリシーと seccomp プロファイルを管理します。詳細は、Security Profiles Operator の有効化 を参照してください。

seccomp および SELinux プロファイルを作成し、ポリシーを Pod にバインドし、ワークロードを記録し、namespace 内のすべてのワーカーノードを同期できます。

Advanced Security Profile Operator タスク を使用して、ログエンリッチャーを有効にしたり、Webhook とメトリックを設定したり、プロファイルを単一の namespace に制限したりします。

必要に応じて Security Profiles Operator のトラブルシューティング を行うか、Red Hat サポート を利用してください。

Operator を削除する前にプロファイルを削除することにより、Security Profiles Operator をアンインストール できます。

7.2. SECURITY PROFILES OPERATOR リリースノート

Security Profiles Operator は、セキュアコンピューティング (seccomp) と SELinux プロファイルをカスタムリソースとして定義し、特定の namespace 内のすべてのノードにプロファイルを同期する方法を提供します。

以下のリリースノートは、OpenShift Container Platform の Security Profiles Operator の開発履歴を記録したものです。

Security Profiles Operator の概要については、Security Profiles Operator の概要 を参照してください。

7.2.1. Security Profiles Operator 0.9.0

Security Profiles Operator 0.9.0 に関するアドバイザリー RHBA-2025:15655 - OpenShift Security Profiles Operator update が利用可能です。

この更新では、セキュリティープロファイルが namespace リソースではなくクラスター全体のリソースとして管理されます。Security Profiles Operator を 0.8.6 以降のバージョンに更新するには、手動で移行する必要があります。移行手順は、Security Profiles Operator 0.9.0 Update Migration Guide を参照してください。

7.2.1.1. バグ修正

- この更新前は、semanage 設定ファイルの解析エラーが原因で、spod Pod が起動に失敗し、**CrashLoopBackOff** 状態になることがありました。この問題は、OpenShift Container Platform 4.19 以降の RHEL 9 イメージの命名規則の変更によって発生します。(**OCPBUGS-55829**)

- この更新前は、リコンサイラータイプの不一致エラーが原因で、Security Profiles Operator が新しく追加されたノードに **RawSelinuxProfile** を適用できませんでした。この更新により、Operator は **RawSelinuxProfile** オブジェクトを正しく処理するようになり、ポリシーが期待どおりにすべてのノードに適用されるようになりました。(**OCPBUGS-33718**)

7.2.2. Security Profiles Operator 0.8.6

Security Profiles Operator 0.8.6 については、次のアドバイザリーが利用できます。

- RHBA-2024:10380 - OpenShift Security Profiles Operator update

この更新には、基礎となるベースイメージの依存関係のアップグレードが含まれています。

7.2.3. Security Profiles Operator 0.8.5

Security Profiles Operator 0.8.5 については、次のアドバイザリーが利用できます。

- RHBA-2024:5016 - OpenShift Security Profiles Operator のバグ修正更新

7.2.3.1. バグ修正

- Web コンソールから Security Profile Operator をインストールしようとすると、Operator 推奨のクラスターモニタリングを有効にするオプションが namespace で使用できませんでした。この更新により、namespace で Operator 推奨のクラスターモニタリングを有効にできるようになりました。(**OCPBUGS-37794**)
- 以前は、Security Profiles Operator が OperatorHub に断続的に表示されず、Web コンソール経由で Operator をインストールするためのアクセスが制限されていました。この更新により、Security Profiles Operator が OperatorHub に存在します。

7.2.4. Security Profiles Operator 0.8.4

Security Profiles Operator 0.8.4 には、次のアドバイザリーを利用できます。

- RHBA-2024:4781 - OpenShift Security Profiles Operator bug fix update

この更新は、基になる依存関係の CVE に対処します。

7.2.4.1. 新機能および機能拡張

- ワイルドカードを設定することで、**ProfileBinding** オブジェクトの **image** 属性にデフォルトのセキュリティープロファイルを指定できるようになりました。詳細は、ProfileBindings を使用してワークロードをプロファイルにバインドする (SELinux) および ProfileBindings を使用してワークロードをプロファイルにバインドする (Seccomp) を参照してください。

7.2.5. Security Profiles Operator 0.8.2

Security Profiles Operator 0.8.2 には、次のアドバイザリーを利用できます。

- RHBA-2023:5958 - OpenShift Security Profiles Operator bug fix update

7.2.5.1. バグ修正

- これまで、**SELinuxProfile** オブジェクトは同じ namespace からカスタム属性を継承しませんでした。今回の更新でこの問題は解決され、**SELinuxProfile** オブジェクト属性が期待どおりに同じ namespace から継承されるようになりました。(OCPBUGS-17164)

- 以前は、RawSELinuxProfiles の作成プロセス中にハングし、**Installed** 状態に到達しませんでした。今回の更新でこの問題は解決され、RawSELinuxProfiles が正常に作成されるようになりました。(OCPBUGS-19744)

- 以前は、パッチを適用して **enableLogEnricher** を **true** にすると、**seccompProfile log-enricher-trace** Pod が **Pending** 状態でスタックしていました。今回の更新では、**log-enricher-trace** Pod の状態は期待どおりに **Installed** になります。(OCPBUGS-22182)

- 以前は、Security Profiles Operator は高カーディナリティーメトリクスを生成し、そのため Prometheus Pod が大量のメモリーを使用していました。今回の更新により、次のメトリクスは Security Profiles Operator namespace に適用されなくなります。
  - **rest_client_request_duration_seconds**
  - **rest_client_request_size_bytes**
  - **rest_client_response_size_bytes**
    (OCPBUGS-22406)

7.2.6. Security Profiles Operator 0.8.0

Security Profiles Operator 0.8.0 には、次のアドバイザリーを利用できます。

- RHBA-2023:4689 - OpenShift Security Profiles Operator bug fix update

7.2.6.1. バグ修正

- 以前は、非接続クラスターに Security Profiles Operator をインストールしようとすると、SHA による再ラベル付が問題となり、セキュアなハッシュが間違っていました。この更新により、提供された SHA は非接続環境でも一貫して動作するようになりました。(OCPBUGS-14404)

7.2.7. Security Profiles Operator 0.7.1

Security Profiles Operator 0.7.1 には、次のアドバイザリーを利用できます。

- RHSA-2023:2029 - OpenShift Security Profiles Operator バグ修正の更新

7.2.7.1. 新機能および機能拡張

- Security Profiles Operator (SPO) は、RHEL 8 および 9 ベースの RHCOS システムに適切な **selinuxd** イメージを自動的に選択するようになりました。

  重要

  非接続環境のイメージをミラーリングするユーザーは、Security Profiles Operator によって提供される両方の **selinuxd** イメージをミラーリングする必要があります。

- **spod** デーモン内でメモリー最適化を有効にできるようになりました。詳細は、spod デーモンでのメモリー最適化の有効化 を参照してください。

> **注記**
>
> デフォルトで SPO メモリーの最適化は有効になっていません。

- デーモンリソース要件を設定できるようになりました。詳細は、デーモンリソース要件のカスタマイズ を参照してください。
- 優先クラス名を **spod** 設定で指定できるようになりました。詳細は、spod デーモン Pod のカスタム優先クラス名の設定 を参照してください。

7.2.7.2. 非推奨の機能および削除された機能

- デフォルトの **nginx-1.19.1** seccomp プロファイルが Security Profiles Operator デプロイメントから削除されました。

7.2.7.3. バグ修正

- これまで、Security Profiles Operator (SPO) SELinux ポリシーはコンテナーテンプレートから低レベルのポリシー定義を継承しませんでした。net_container などの別のテンプレートを選択した場合、コンテナーテンプレートにのみ存在する低レベルのポリシー定義が必要となるため、ポリシーは機能しません。この問題は、SPO SELinux ポリシーが SELinux ポリシーを SPO カスタム形式から共通中間言語 (CIL) 形式に変換しようとすると発生しました。今回の更新により、SPO から CIL への変換を必要とする SELinux ポリシーにコンテナーテンプレートが追加されます。さらに、SPO SELinux ポリシーは、サポートされている任意のポリシーテンプレートから低レベルのポリシー定義を継承できます。(**OCPBUGS-12879**)

7.2.7.4. 既知の問題

- Security Profiles Operator をアンインストールする場合、**MutatingWebhookConfiguration** オブジェクトは削除されないため、手動で削除する必要があります。回避策として、Security Profiles Operator をアンインストールした後に **MutatingWebhookConfiguration** オブジェクトを削除します。これらの手順は、Security Profiles Operator のアンインストール で定義されています。(**OCPBUGS-4687**)

7.2.8. Security Profiles Operator 0.5.2

Security Profiles Operator 0.5.2 には、次のアドバイザリーを利用できます。

- RHBA-2023:0788 - OpenShift Security Profiles Operator バグ修正の更新

この更新は、基礎となる依存関係の CVE に対処するものです。

7.2.8.1. 既知の問題

- Security Profiles Operator をアンインストールする場合、**MutatingWebhookConfiguration** オブジェクトは削除されないため、手動で削除する必要があります。回避策として、Security Profiles Operator をアンインストールした後に **MutatingWebhookConfiguration** オブジェクトを削除します。これらの手順は、Security Profiles Operator のアンインストール で定義されています。(**OCPBUGS-4687**)

7.2.9. Security Profiles Operator 0.5.0

Security Profiles Operator 0.5.0 には、次のアドバイザリーを利用できます。

- RHBA-2022:8762 - OpenShift Security Profiles Operator バグ修正の更新

7.2.9.1. 既知の問題

- Security Profiles Operator をアンインストールする場合、**MutatingWebhookConfiguration** オブジェクトは削除されないため、手動で削除する必要があります。回避策として、Security Profiles Operator をアンインストールした後に **MutatingWebhookConfiguration** オブジェクトを削除します。これらの手順は、Security Profiles Operator のアンインストール で定義されています。(**OCPBUGS-4687**)

7.3. SECURITY PROFILES OPERATOR のサポート

7.3.1. Security Profiles Operator のライフサイクル

Security Profiles Operator は "Rolling Stream" Operator で、OpenShift Container Platform リリースの更新が非同期で利用可能であることを意味します。詳細は、Red Hat カスタマーポータルの OpenShift Operator ライフサイクル を参照してください。

7.3.2. サポート

このドキュメントで説明されている手順、または OpenShift Container Platform 全般で問題が発生した場合は、Red Hat カスタマーポータル にアクセスしてください。

カスタマーポータルでは、次のことができます。

- Red Hat 製品に関するアーティクルおよびソリューションを対象とした Red Hat ナレッジベースの検索またはブラウズ。
- Red Hat サポートに対するサポートケースの送信。
- その他の製品ドキュメントへのアクセス。

クラスターの問題を特定するには、OpenShift Cluster Manager で Insights を使用できます。Insights により、問題の詳細と、利用可能な場合は問題の解決方法に関する情報が提供されます。

このドキュメントを改善するための提案がある場合、またはエラーを見つけた場合は、最も関連性の高いドキュメントコンポーネントについて Jira 課題 を送信してください。セクション名や OpenShift Container Platform バージョンなどの具体的な情報を提供してください。

7.4. SECURITY PROFILES OPERATOR について

OpenShift Container Platform 管理者は、Security Profiles Operator を使用して、クラスターで強化されたセキュリティー対策を定義できます。

重要

Security Profiles Operator は、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみをサポートします。Red Hat Enterprise Linux (RHEL) ノードはサポートされていません。

7.4.1. セキュリティープロファイルについて

セキュリティープロファイルは、クラスター内のコンテナーレベルでセキュリティーを強化できます。

Seccomp セキュリティープロファイルには、プロセスが実行できるシステムコールがリストされています。権限は SELinux よりも広く、ユーザーは **write** などのシステム全体の操作を制限できます。

SELinux セキュリティープロファイルは、システム内のプロセス、アプリケーション、またはファイルへのアクセスと使用を制限するラベルベースのシステムを提供します。環境内のすべてのファイルには、権限を定義するラベルがあります。SELinux プロファイルは、ディレクトリーなどの特定の構造内でアクセスを定義できます。

7.5. SECURITY PROFILES OPERATOR の有効化

Security Profiles Operator を使用する前に、Operator がクラスターにデプロイされていることを確認する必要があります。

重要

この Operator を適切に機能させるには、すべてのクラスターノードのリリースバージョンが同じである必要があります。たとえば、RHCOS を実行しているノードの場合、すべてのノードの RHCOS バージョンが同じである必要があります。

重要

Security Profiles Operator は、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみをサポートします。Red Hat Enterprise Linux (RHEL) ノードはサポートされていません。

重要

Security Profiles Operator は、**x86_64** アーキテクチャーのみをサポートします。

7.5.1. Security Profiles Operator のインストール

前提条件

- **admin** 権限がある。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. Security Profiles Operator を検索し、**Install** をクリックします。
3. Operator が **openshift-security-profiles** namespace にインストールされるように、**Installation mode** と **namespace** のデフォルトの選択を維持します。
4. **Install** をクリックします。

検証

インストールが正常に行われたことを確認するには、以下を実行します。

1. **Operators → Installed Operators** ページに移動します。

2. Security Profiles Operator が **openshift-security-profiles** namespace にインストールされ、そのステータスが **Succeeded** であることを確認します。

Operator が正常にインストールされていない場合、以下を実行します。

1. **Operators → Installed Operators** ページに移動し、**Status** 列でエラーまたは失敗の有無を確認します。

2. **Workloads → Pods** ページに移動し、**openshift-security-profiles** プロジェクトの Pod で問題を報告しているログの有無を確認します。

7.5.2. CLI を使用した Security Profiles Operator のインストール

前提条件

- **admin** 権限がある。

手順

1. **Namespace** オブジェクトを定義します。

   namespace-object.yaml の例

   ```
   apiVersion: v1
   kind: Namespace
   metadata:
       name: openshift-security-profiles
   labels:
     openshift.io/cluster-monitoring: "true"
   ```

2. **Namespace** オブジェクトを作成します。

   ```
   $ oc create -f namespace-object.yaml
   ```

3. **OperatorGroup** オブジェクトを定義します。

   operator-group-object.yaml の例

   ```
   apiVersion: operators.coreos.com/v1
   kind: OperatorGroup
   metadata:
     name: security-profiles-operator
     namespace: openshift-security-profiles
   ```

4. **OperatorGroup** オブジェクトを作成します。

   ```
   $ oc create -f operator-group-object.yaml
   ```

5. **Subscription** オブジェクトを定義します。

   subscription-object.yaml の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: security-profiles-operator-sub
  namespace: openshift-security-profiles
spec:
  channel: release-alpha-rhel-8
  installPlanApproval: Automatic
  name: security-profiles-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
```

6. **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription-object.yaml
```

> **注記**
>
> グローバルスケジューラー機能を設定して **defaultNodeSelector** を有効にする場合、namespace を手動で作成し、**openshift-security-profiles** namespace のアノテーション、または Security Profiles Operator がインストールされた namespace を **openshift.io/node-selector: “”** で更新する必要があります。これにより、デフォルトのノードセレクターが削除され、デプロイメントの失敗を防ぐことができます。

検証

1. 次の CSV ファイルを調べて、インストールが成功したことを確認します。

```
$ oc get csv -n openshift-security-profiles
```

2. 次のコマンドを実行して、Security Profiles Operator が動作していることを確認します。

```
$ oc get deploy -n openshift-security-profiles
```

7.5.3. ログの詳細度の設定

Security Profiles Operator は、デフォルトのロギング詳細度 **0** と拡張詳細度 **1** をサポートします。

手順

- 強化されたロギングの詳細度を有効にするには、**spod** 設定にパッチを適用し、次のコマンドを実行して値を調整します。

```
$ oc -n openshift-security-profiles patch spod \
    spod --type=merge -p '{"spec":{"verbosity":1}}'
```

出力例

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

7.6. SECCOMP プロファイルの管理

seccomp プロファイルを作成および管理し、それらをワークロードにバインドします。

重要

Security Profiles Operator は、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみをサポートします。Red Hat Enterprise Linux (RHEL) ノードはサポートされていません。

7.6.1. seccomp プロファイルの作成

プロファイルを作成するには、**SeccompProfile** オブジェクトを使用します。

SeccompProfile オブジェクトは、コンテナー内のシステムコールを制限して、アプリケーションのアクセスを制限できます。

手順

1. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project my-namespace
```

2. **SeccompProfile** オブジェクトを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  name: profile1
spec:
  defaultAction: SCMP_ACT_LOG
```

seccomp プロファイルは **/var/lib/kubelet/seccomp/operator/<namespace>/<name>.json** に保存されます。

init コンテナーは Security Profiles Operator のルートディレクトリーを作成し、 **root** グループまたはユーザー ID 権限なしで Operator を実行します。ルートレスプロファイルストレージ **/var/lib/openshift-security-profiles** から、kubelet root **/var/lib/kubelet/seccomp/operator** 内のデフォルトの **seccomp** root パスへのシンボリックリンクが作成されます。

7.6.2. Pod への seccomp プロファイルの適用

Pod を作成して、作成したプロファイルの 1 つを適用します。

手順

1. **securityContext** を定義する Pod オブジェクトを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  securityContext:
    runAsNonRoot: true
```

```
    seccompProfile:
      type: Localhost
      localhostProfile: operator/profile1.json
  containers:
    - name: test-container
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

2. 次のコマンドを実行して、**seccompProfile.localhostProfile** 属性のプロファイルパスを表示します。

```
$ oc get seccompprofile profile1 --output wide
```

出力例

```
NAME       STATUS     AGE   SECCOMPPROFILE.LOCALHOSTPROFILE
profile1   Installed  14s   operator/profile1.json
```

3. 次のコマンドを実行して、localhost プロファイルへのパスを表示します。

```
$ oc get sp profile1 --output=jsonpath='{.status.localhostProfile}'
```

出力例

```
operator/profile1.json
```

4. **localhostProfile** の出力をパッチファイルに適用します。

```
spec:
  template:
    spec:
      securityContext:
        seccompProfile:
          type: Localhost
          localhostProfile: operator/profile1.json
```

5. 次のコマンドを実行して、プロファイルを他のワークロード (**Deployment** オブジェクトなど) に適用します。

```
$ oc -n my-namespace patch deployment myapp --patch-file patch.yaml --type=merge
```

出力例

```
deployment.apps/myapp patched
```

検証

- 次のコマンドを実行して、プロファイルが正しく適用されたことを確認します。

```
$ oc -n my-namespace get deployment myapp --
output=jsonpath='{.spec.template.spec.securityContext}' | jq .
```

出力例

```
{
  "seccompProfile": {
    "localhostProfile": "operator/profile1.json",
    "type": "localhost"
  }
}
```

7.6.2.1. ProfileBindings を使用してワークロードをプロファイルにバインドする

ProfileBinding リソースを使用して、セキュリティープロファイルをコンテナーの **SecurityContext** にバインドできます。

手順

1. **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** イメージを使用する Pod をサンプルの **SeccompProfile** プロファイルにバインドするには、Pod と **SeccompProfile** オブジェクトと同じ namespace に **ProfileBinding** オブジェクトを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileBinding
metadata:
  namespace: my-namespace
  name: nginx-binding
spec:
  profileRef:
    kind: SeccompProfile (1)
    name: profile (2)
  image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21 (3)
```

(1) **kind:** 変数はプロファイルの種類を参照します。

(2) **name:** 変数は、プロファイルの名前を参照します。

(3) イメージ属性にワイルドカードを使用する (**image: "*"**) と、デフォルトのセキュリティープロファイルを有効にすることができます。

重要

image: "*" ワイルドカード属性を使用すると、すべての新しい Pod が、指定された namespace 内のデフォルトのセキュリティープロファイルにバインドされます。

2. 次のコマンドを実行して、namespace に **enable-binding=true** のラベルを付けます。

```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

3. **test-pod.yaml** という名前の Pod を定義します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

4. Pod を作成します。

```
$ oc create -f test-pod.yaml
```

注記

Pod がすでに存在する場合、Pod を再作成しなければバインディングは適切に機能しません。

検証

- 次のコマンドを実行して、Pod が **ProfileBinding** を継承していることを確認します。

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seccompProfile}'
```

出力例

```
{"localhostProfile":"operator/profile.json","type":"Localhost"}
```

7.6.3. ワークロードからのプロファイルの記録

Security Profiles Operator は、**ProfileRecording** オブジェクトを使用してシステムコールを記録できるため、アプリケーションのベースラインプロファイルを簡単に作成できます。

ログエンリッチャーを使用して seccomp プロファイルを記録する場合は、ログエンリッチャー機能が有効になっていることを確認します。詳細は、**関連情報** を参照してください。

注記

privileged: true のセキュリティーコンテキスト制限を持つコンテナーにより、ログベースの記録が防止されます。特権コンテナーは seccomp ポリシーの対象ではなく、ログベースの記録では特別な seccomp プロファイルを使用してイベントを記録します。

手順

1. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project my-namespace
```

2. 次のコマンドを実行して、namespace に **enable-recording=true** のラベルを付けます。

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. **recorder: logs** 変数を含む **ProfileRecording** オブジェクトを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  namespace: my-namespace
  name: test-recording
spec:
  kind: SeccompProfile
  recorder: logs
  podSelector:
    matchLabels:
      app: my-app
```

4. 記録するワークロードを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
  name: my-pod
  labels:
    app: my-app
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

5. 次のコマンドを入力して、Pod が **Running** 状態であることを確認します。

```
$ oc -n my-namespace get pods
```

出力例

```
NAME     READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

–

6. エンリッチャーがそれらのコンテナーの監査ログを受信することを示していることを確認します。

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

出力例

```
I0523 14:19:08.747313  430694 enricher.go:445] log-enricher "msg"="audit"
"container"="redis" "executable"="/usr/local/bin/redis-server" "namespace"="my-namespace"
"node"="xiyuan-23-5g2q9-worker-eastus2-6rpgf" "pid"=656802 "pod"="my-pod" "syscallID"=0
"syscallName"="read" "timestamp"="1684851548.745:207179" "type"="seccomp"
```

検証

1. Pod を削除します。

```
$ oc -n my-namespace delete pod my-pod
```

2. Security Profiles Operator が 2 つの seccomp プロファイルを調整することを確認します。

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording
```

seccompprofile の出力例

```
NAME                  STATUS      AGE
test-recording-nginx   Installed   2m48s
test-recording-redis   Installed   2m48s
```

7.6.3.1. コンテナーごとのプロファイルインスタンスのマージ

デフォルトでは、各コンテナーインスタンスは個別のプロファイルに記録されます。Security Profiles Operator は、コンテナーごとのプロファイルを 1 つのプロファイルにマージできます。プロファイルのマージは、**ReplicaSet** または **Deployment** オブジェクトを使用してアプリケーションをデプロイするときに役立ちます。

手順

1. **ProfileRecording** オブジェクトを編集して、**mergeStrategy: containers** 変数を含めます。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  # The name of the Recording is the same as the resulting SeccompProfile CRD
  # after reconciliation.
  name: test-recording
  namespace: my-namespace
spec:
  kind: SeccompProfile
  recorder: logs
  mergeStrategy: containers
  podSelector:
    matchLabels:
```

```
      app: sp-record
```

2. 以下のコマンドを実行して namespace にラベルを付けます。

```
$ oc label ns my-namespace security.openshift.io/scc.podSecurityLabelSync=false pod-security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-security.kubernetes.io/warn=privileged --overwrite=true
```

3. 次の YAML を使用してワークロードを作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sp-record
  template:
    metadata:
      labels:
        app: sp-record
    spec:
      serviceAccountName: spo-record-sa
      containers:
      - name: nginx-record
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

4. 個々のプロファイルを記録するには、次のコマンドを実行してデプロイメントを削除します。

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. プロファイルをマージするには、次のコマンドを実行してプロファイルの記録を削除します。

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. マージ操作を開始して結果プロファイルを生成するには、次のコマンドを実行します。

```
$ oc get seccompprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

seccompprofiles の出力例

```
NAME                          STATUS      AGE
test-recording-nginx-record   Installed   55s
```

7. いずれかのコンテナーで使用されている権限を表示するには、次のコマンドを実行します。

```
$ oc get seccompprofiles test-recording-nginx-record -o yaml
```

7.6.4. 関連情報

- Security Context Constraints の管理
- OpenShift での SCC の管理
- ログエンリッチャーの使用
- セキュリティープロファイルについて

7.7. SELINUX プロファイルの管理

SELinux プロファイルを作成および管理し、それらをワークロードにバインドします。

重要

Security Profiles Operator は、Red Hat Enterprise Linux CoreOS (RHCOS) ワーカーノードのみをサポートします。Red Hat Enterprise Linux (RHEL) ノードはサポートされていません。

7.7.1. SELinux プロファイルの作成

SelinuxProfile オブジェクトを使用してプロファイルを作成します。

SelinuxProfile オブジェクトには、セキュリティー強化と読みやすさを向上させるいくつかの機能があります。

- 継承するプロファイルを現在の namespace またはシステム全体のプロファイルに制限します。通常、システムには多くのプロファイルがインストールされていますが、クラスターワークロードではサブセットのみを使用する必要があるため、継承可能なシステムプロファイルは、**spec.selinuxOptions.allowedSystemProfiles** の **spod** インスタンスにリストされています。
- 権限、クラス、およびラベルの基本的な検証を実行します。
- ポリシーを使用するプロセスを説明する新しいキーワード **@self** を追加します。これにより、ポリシーの使用は名前と namespace に基づいているため、ワークロードと namespace の間でポリシーを簡単に再利用できます。
- SELinux CIL 言語で直接プロファイルを作成する場合と比較して、セキュリティーを強化し、読みやすくするための機能を追加します。

手順

1. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project nginx-deploy
```

2. 次の **SelinuxProfile** オブジェクトを作成して、特権のないワークロードで使用できるポリシーを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha2
kind: SelinuxProfile
metadata:
```

```
  name: nginx-secure
spec:
  allow:
    '@self':
      tcp_socket:
      - listen
    http_cache_port_t:
      tcp_socket:
      - name_bind
    node_t:
      tcp_socket:
      - node_bind
  inherit:
  - kind: System
    name: container
```

3. 次のコマンドを実行して、**selinuxd** がポリシーをインストールするのを待ちます。

```
$ oc wait --for=condition=ready selinuxprofile nginx-secure
```

出力例

```
selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure condition met
```

ポリシーは、Security Profiles Operator が所有するコンテナー内の **emptyDir** に配置されます。ポリシーは Common Intermediate Language (CIL) 形式で **/etc/selinux.d/<name>_<namespace>.cil** に保存されます。

4. 以下のコマンドを実行して Pod を作成します。

```
$ oc -n openshift-security-profiles rsh -c selinuxd ds/spod
```

検証

1. 次のコマンドを実行して、**cat** でファイルの内容を表示します。

```
$ cat /etc/selinux.d/nginx-secure_.cil
```

出力例

```
(block nginx-secure_
(blockinherit container)
(allow process nginx-secure_.process ( tcp_socket ( listen )))
(allow process http_cache_port_t ( tcp_socket ( name_bind )))
(allow process node_t ( tcp_socket ( node_bind )))
)
```

2. 次のコマンドを実行して、ポリシーがインストールされていることを確認します。

```
$ semodule -l | grep nginx-secure
```

出力例

```
nginx-secure_
```

7.7.2. Pod への SELinux プロファイルの適用

Pod を作成して、作成したプロファイルの 1 つを適用します。

SELinux プロファイルの場合、namespace にラベルを付けて、特権 ワークロードを許可する必要があります。

手順

1. 次のコマンドを実行して、**scc.podSecurityLabelSync=false** ラベルを **nginx-deploy** namespace に適用します。

```
$ oc label ns nginx-deploy security.openshift.io/scc.podSecurityLabelSync=false
```

2. 次のコマンドを実行して、**privileged** ラベルを **nginx-deploy** namespace に適用します。

```
$ oc label ns nginx-deploy --overwrite=true pod-security.kubernetes.io/enforce=privileged
```

3. 次のコマンドを実行して、SELinux プロファイルの使用文字列を取得します。

```
$ oc get selinuxprofile.security-profiles-operator.x-k8s.io/nginx-secure -
ojsonpath='{.status.usage}'
```

出力例

```
nginx-secure_.process
```

4. ワークロードマニフェストの出力文字列を **.spec.containers[].securityContext.seLinuxOptions** 属性に適用します。

```
apiVersion: v1
kind: Pod
metadata:
  name: nginx-secure
  namespace: nginx-deploy
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - image: nginxinc/nginx-unprivileged:1.21
      name: nginx
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
        seLinuxOptions:
          # NOTE: This uses an appropriate SELinux type
          type: nginx-secure_.process
```

重要

ワークロードを作成する前に、SELinux **type** が存在している必要があります。

7.7.2.1. SELinux ログポリシーの適用

ポリシー違反または AVC 拒否をログに記録するには、**SElinuxProfile** プロファイルを **permissive** に設定します。

重要

この手順では、ロギングポリシーを定義します。施行ポリシーを設定しません。

手順

- **permissive: true** を **SElinuxProfile** に追加します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha2
kind: SelinuxProfile
metadata:
  name: nginx-secure
spec:
  permissive: true
```

7.7.2.2. ProfileBindings を使用してワークロードをプロファイルにバインドする

ProfileBinding リソースを使用して、セキュリティープロファイルをコンテナーの **SecurityContext** にバインドできます。

手順

1. **quay.io/security-profiles-operator/test-nginx-unprivileged:1.21** イメージを使用する Pod をサンプルの **SelinuxProfile** プロファイルにバインドするには、Pod と **SelinuxProfile** オブジェクトと同じ namespace に **ProfileBinding** オブジェクトを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileBinding
metadata:
  namespace: my-namespace
  name: nginx-binding
spec:
  profileRef:
    kind: SelinuxProfile 1
    name: profile 2
  image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21 3
```

1 **kind:** 変数はプロファイルの種類を参照します。

2 **name:** 変数は、プロファイルの名前を参照します。

3 イメージ属性にワイルドカードを使用する (**image: "*"**) と、デフォルトのセキュリティープロファイルを有効にすることができます。

重要

image: "*" ワイルドカード属性を使用すると、すべての新しい Pod が、指定された namespace 内のデフォルトのセキュリティープロファイルにバインドされます。

2. 次のコマンドを実行して、namespace に **enable-binding=true** のラベルを付けます。

```
$ oc label ns my-namespace spo.x-k8s.io/enable-binding=true
```

3. **test-pod.yaml** という名前の Pod を定義します。

```
apiVersion: v1
kind: Pod
metadata:
  name: test-pod
spec:
  containers:
  - name: test-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
```

4. Pod を作成します。

```
$ oc create -f test-pod.yaml
```

注記

Pod がすでに存在する場合、Pod を再作成しなければバインディングは適切に機能しません。

検証

- 次のコマンドを実行して、Pod が **ProfileBinding** を継承していることを確認します。

```
$ oc get pod test-pod -o jsonpath='{.spec.containers[*].securityContext.seLinuxOptions.type}'
```

出力例

```
profile_.process
```

7.7.2.3. コントローラーと SecurityContextConstraints の複製

デプロイメントやデーモンセットなど、コントローラーをレプリケートするための SELinux ポリシーをデプロイする場合、コントローラーによって生成された **Pod** オブジェクトは、ワークロードを作成するユーザーの ID で実行されないことに注意してください。**ServiceAccount** が選択されていない場合、Pod は、カスタムセキュリティーポリシーの使用を許可しない、制限された **SecurityContextConstraints** (SCC) の使用に戻る可能性があります。

手順

1. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project nginx-secure
```

2. 次の **RoleBinding** オブジェクトを作成して、SELinux ポリシーを **nginx-secure** namespace で使用できるようにします。

```yaml
kind: RoleBinding
apiVersion: rbac.authorization.k8s.io/v1
metadata:
  name: spo-nginx
  namespace: nginx-secure
subjects:
- kind: ServiceAccount
  name: spo-deploy-test
roleRef:
  kind: Role
  name: spo-nginx
  apiGroup: rbac.authorization.k8s.io
```

3. **Role** オブジェクトを作成します。

```yaml
apiVersion: rbac.authorization.k8s.io/v1
kind: Role
metadata:
  creationTimestamp: null
  name: spo-nginx
  namespace: nginx-secure
rules:
- apiGroups:
  - security.openshift.io
  resources:
  - securitycontextconstraints
  resourceNames:
  - privileged
  verbs:
  - use
```

4. **ServiceAccount** オブジェクトを作成します。

```yaml
apiVersion: v1
kind: ServiceAccount
metadata:
  creationTimestamp: null
  name: spo-deploy-test
  namespace: nginx-secure
```

5. **Deployment** オブジェクトを作成します。

```yaml
apiVersion: apps/v1
kind: Deployment
metadata:
  name: selinux-test
  namespace: nginx-secure
  metadata:
    labels:
```

```
    app: selinux-test
spec:
  replicas: 3
  selector:
    matchLabels:
      app: selinux-test
  template:
    metadata:
      labels:
        app: selinux-test
    spec:
      serviceAccountName: spo-deploy-test
      securityContext:
        seLinuxOptions:
          type: nginx-secure_.process (1)
      containers:
      - name: nginx-unpriv
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

(1) デプロイメントが作成される前に、**.seLinuxOptions.type** が存在している必要があります。

> **注記**
>
> SELinux タイプはワークロードで指定されておらず、SCC によって処理されます。デプロイと **ReplicaSet** によって Pod が作成されると、Pod は適切なプロファイルで実行されます。

SCC が、正しいサービスアカウントのみで使用できることを確認してください。詳細は、**その他のリソース** を参照してください。

7.7.3. ワークロードからのプロファイルの記録

Security Profiles Operator は、**ProfileRecording** オブジェクトを使用してシステムコールを記録できるため、アプリケーションのベースラインプロファイルを簡単に作成できます。

ログエンリッチャーを使用して SELinux プロファイルを記録する場合は、ログエンリッチャー機能が有効になっていることを確認します。詳細は、**関連情報** を参照してください。

> **注記**
>
> **privileged: true** のセキュリティーコンテキスト制限を持つコンテナーにより、ログベースの記録が防止されます。特権コンテナーは SELinux ポリシーの対象ではなく、ログベースの記録では特別な SELinux プロファイルを使用してイベントを記録します。

手順

1. 次のコマンドを実行してプロジェクトを作成します。

```
$ oc new-project my-namespace
```

2. 次のコマンドを実行して、namespace に **enable-recording=true** のラベルを付けます。

```
$ oc label ns my-namespace spo.x-k8s.io/enable-recording=true
```

3. **recorder: logs** 変数を含む **ProfileRecording** オブジェクトを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  namespace: my-namespace
  name: test-recording
spec:
  kind: SelinuxProfile
  recorder: logs
  podSelector:
    matchLabels:
      app: my-app
```

4. 記録するワークロードを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  namespace: my-namespace
  name: my-pod
  labels:
    app: my-app
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: RuntimeDefault
  containers:
    - name: nginx
      image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
      ports:
        - containerPort: 8080
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
    - name: redis
      image: quay.io/security-profiles-operator/redis:6.2.1
      securityContext:
        allowPrivilegeEscalation: false
        capabilities:
          drop: [ALL]
```

5. 次のコマンドを入力して、Pod が **Running** 状態であることを確認します。

```
$ oc -n my-namespace get pods
```

出力例

```
NAME     READY   STATUS    RESTARTS   AGE
my-pod   2/2     Running   0          18s
```

6. エンリッチャーがそれらのコンテナーの監査ログを受信することを示していることを確認します。

```
$ oc -n openshift-security-profiles logs --since=1m --selector name=spod -c log-enricher
```

出力例

```
I0517 13:55:36.383187  348295 enricher.go:376] log-enricher "msg"="audit"
"container"="redis" "namespace"="my-namespace" "node"="ip-10-0-189-53.us-east-
2.compute.internal" "perm"="name_bind" "pod"="my-pod" "profile"="test-
recording_redis_6kmrb_1684331729"
"scontext"="system_u:system_r:selinuxrecording.process:s0:c4,c27" "tclass"="tcp_socket"
"tcontext"="system_u:object_r:redis_port_t:s0" "timestamp"="1684331735.105:273965"
"type"="selinux"
```

検証

1. Pod を削除します。

```
$ oc -n my-namespace delete pod my-pod
```

2. Security Profiles Operator が 2 つの SELinux プロファイルを調整することを確認します。

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording
```

selinuxprofile の出力例

```
NAME                  USAGE                          STATE
test-recording-nginx   test-recording-nginx_.process   Installed
test-recording-redis   test-recording-redis_.process   Installed
```

7.7.3.1. コンテナーごとのプロファイルインスタンスのマージ

デフォルトでは、各コンテナーインスタンスは個別のプロファイルに記録されます。Security Profiles Operator は、コンテナーごとのプロファイルを 1 つのプロファイルにマージできます。プロファイルのマージは、**ReplicaSet** または **Deployment** オブジェクトを使用してアプリケーションをデプロイするときに役立ちます。

手順

1. **ProfileRecording** オブジェクトを編集して、**mergeStrategy: containers** 変数を含めます。

```
apiVersion: security-profiles-operator.x-k8s.io/v1alpha1
kind: ProfileRecording
metadata:
  # The name of the Recording is the same as the resulting SelinuxProfile CRD
  # after reconciliation.
  name: test-recording
  namespace: my-namespace
```

```
spec:
  kind: SelinuxProfile
  recorder: logs
  mergeStrategy: containers
  podSelector:
    matchLabels:
      app: sp-record
```

2. 以下のコマンドを実行して namespace にラベルを付けます。

```
$ oc label ns my-namespace security.openshift.io/scc.podSecurityLabelSync=false pod-security.kubernetes.io/enforce=privileged pod-security.kubernetes.io/audit=privileged pod-security.kubernetes.io/warn=privileged --overwrite=true
```

3. 次の YAML を使用してワークロードを作成します。

```
apiVersion: apps/v1
kind: Deployment
metadata:
  name: nginx-deploy
  namespace: my-namespace
spec:
  replicas: 3
  selector:
    matchLabels:
      app: sp-record
  template:
    metadata:
      labels:
        app: sp-record
    spec:
      serviceAccountName: spo-record-sa
      containers:
      - name: nginx-record
        image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
        ports:
        - containerPort: 8080
```

4. 個々のプロファイルを記録するには、次のコマンドを実行してデプロイメントを削除します。

```
$ oc delete deployment nginx-deploy -n my-namespace
```

5. プロファイルをマージするには、次のコマンドを実行してプロファイルの記録を削除します。

```
$ oc delete profilerecording test-recording -n my-namespace
```

6. マージ操作を開始して結果プロファイルを生成するには、次のコマンドを実行します。

```
$ oc get selinuxprofiles -lspo.x-k8s.io/recording-id=test-recording -n my-namespace
```

selinuxprofiles の出力例

```
NAME                        USAGE                                STATE
test-recording-nginx-record   test-recording-nginx-record_.process   Installed
```

■

7. いずれかのコンテナーで使用されている権限を表示するには、次のコマンドを実行します。

```
$ oc get selinuxprofiles test-recording-nginx-record -o yaml
```

7.7.3.2. About seLinuxContext: RunAsAny

SELinux ポリシーの記録は、記録される Pod に特別な SELinux タイプを挿入する Webhook で実装されます。SELinux タイプは、Pod を **permissive** モードで実行し、すべての AVC 拒否を **audit.log** に記録します。デフォルトでは、カスタム SELinux ポリシーを使用したワークロードの実行は許可されていませんが、自動生成されたタイプが使用されます。

ワークロードを記録するには、Webhook が許容された SELinux タイプを挿入できる SCC を使用する権限を持つサービスアカウントを使用する必要があります。**privileged** SCC には、 **seLinuxContext: RunAsAny** が含まれています。

さらに、カスタム SELinux ポリシーの使用を許可するのは **privileged** の Pod Security Standard のみであるため、クラスターで Pod Security Admission が有効になっている場合は、namespace に **pod-security.kubernetes.io/enforce: privileged** 付きのラベルを付ける必要があります。

7.7.4. 関連情報

- Security Context Constraints の管理
- OpenShift での SCC の管理
- ログエンリッチャーの使用
- セキュリティープロファイルについて

7.8. SECURITY PROFILES OPERATOR の高度なタスク

高度なタスクを使用して、メトリックを有効にしたり、Webhook を設定したり、syscall を制限したりします。

7.8.1. seccomp プロファイルで許可されるシステムコールを制限する

Security Profiles Operator は、デフォルトでは **seccomp** プロファイル内の **syscalls** を制限しません。**spod** 設定で、許可される **syscalls** のリストを定義できます。

手順

- **allowedSyscalls** のリストを定義するには、次のコマンドを実行して **spec** パラメーターを調整します。

```
$ oc -n openshift-security-profiles patch spod spod --type merge \
    -p '{"spec":{"allowedSyscalls": ["exit", "exit_group", "futex", "nanosleep"]}}'
```

重要

この Operator は **seccomp** プロファイルのみをインストールします。このプロファイルには、許可リストに定義されている **syscalls** の一部が含まれています。このルールセットに準拠していないすべてのプロファイルは拒否されます。

許可される **syscalls** のリストが **spod** 設定で変更されると、Operator はすでにインストールされている非準拠のプロファイルを特定し、それを自動的に削除します。

7.8.2. コンテナーランタイムのベースシステムコール

baseProfileName 属性を使用すると、特定のランタイムがコンテナーを起動するために必要な最小限の **syscalls** を設定できます。

手順

- **SeccompProfile** kind オブジェクトを編集し、**baseProfileName: runc-v1.0.0** を **spec** フィールドに追加します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  name: example-name
spec:
  defaultAction: SCMP_ACT_ERRNO
  baseProfileName: runc-v1.0.0
  syscalls:
    - action: SCMP_ACT_ALLOW
      names:
        - exit_group
```

7.8.3. spod デーモンでのメモリー最適化の有効化

プロファイルの記録が有効になっている場合、**spod** デーモンプロセス内で実行されているコントローラーは、クラスター内の使用可能な Pod をすべて監視します。そこれにより、大規模なクラスターではメモリー使用量が非常に多くなり、その結果、**spod** デーモンがメモリー不足になったり、クラッシュしたりする可能性があります。

クラッシュを防ぐために、プロファイル記録用にラベル付けされた Pod のみをキャッシュメモリーにロードするように **spod** デーモンを設定できます。

注記

デフォルトで SPO メモリーの最適化は有効になっていません。

手順

1. 次のコマンドを実行してメモリー最適化を有効にします。

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"enableMemoryOptimization":true}}'
```

2. Pod のセキュリティープロファイルを記録するには、Pod に **spo.x-k8s.io/enable-recording: "true"** のラベルを付ける必要があります。

```
apiVersion: v1
kind: Pod
metadata:
  name: my-recording-pod
  labels:
    spo.x-k8s.io/enable-recording: "true"
# ...
```

7.8.4. デーモンリソース要件のカスタマイズ

デーモンコンテナーのデフォルトのリソース要件は、**spod** 設定の **daemonResourceRequirements** フィールドを使用して調整できます。

手順

- デーモンコンテナーのメモリーと CPU の要求と制限を指定するには、次のコマンドを実行します。

```
$ oc -n openshift-security-profiles patch spod spod --type merge -p \
    '{"spec":{"daemonResourceRequirements": { \
    "requests": {"memory": "256Mi", "cpu": "250m"}, \
    "limits": {"memory": "512Mi", "cpu": "500m"}}}}'
```

7.8.5. spod デーモン Pod のカスタム優先クラス名の設定

spod デーモン Pod のデフォルトの優先クラス名は **system-node-critical** に設定されます。カスタム優先クラス名は、**spod** 設定の **priorityClassName** フィールドに値を入力して設定できます。

手順

- 次のコマンドを実行して、優先クラス名を設定します。

```
$ oc -n openshift-security-profiles patch spod spod --type=merge -p '{"spec":
{"priorityClassName":"my-priority-class"}}'
```

出力例

```
securityprofilesoperatordaemon.openshift-security-profiles.x-k8s.io/spod patched
```

7.8.6. メトリクスの使用

openshift-security-profiles namespace は、kube-rbac-proxy コンテナーによって保護されるメトリクスエンドポイントを提供します。すべてのメトリクスは、**openshift-security-profiles** namespace 内の **metrics** サービスによって公開されます。

Security Profiles Operator には、クラスター内からメトリックを取得するためのクラスターロールと、対応するバインディング **spo-metrics-client** が含まれています。次の 2 つのメトリクスパスを使用できます。

- **metrics.openshift-security-profiles/metrics**: コントローラーのランタイムメトリック用
- **metrics.openshift-security-profiles/metrics-spod**: Operator デーモンメトリクス用

手順

1. メトリクスサービスのステータスを表示するには、次のコマンドを実行します。

```
$ oc get svc/metrics -n openshift-security-profiles
```

出力例

```
NAME     TYPE       CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
metrics  ClusterIP  10.0.0.228   <none>        443/TCP   43s
```

2. メトリクスを取得するには、次のコマンドを実行して、**openshift-security-profiles** namespace でデフォルトの **ServiceAccount** トークンを使用してサービスエンドポイントをクエリーします。

```
$ oc run --rm -i --restart=Never --image=registry.fedoraproject.org/fedora-minimal:latest \
    -n openshift-security-profiles metrics-test -- bash -c \
    'curl -ks -H "Authorization: Bearer $(cat
/var/run/secrets/kubernetes.io/serviceaccount/token)" https://metrics.openshift-security-
profiles/metrics-spod'
```

出力例

```
# HELP security_profiles_operator_seccomp_profile_total Counter about seccomp profile
operations.
# TYPE security_profiles_operator_seccomp_profile_total counter
security_profiles_operator_seccomp_profile_total{operation="delete"} 1
security_profiles_operator_seccomp_profile_total{operation="update"} 2
```

3. 別の namespace からメトリックを取得するには、次のコマンドを実行して **ServiceAccount** を **spo-metrics-client ClusterRoleBinding** にリンクします。

```
$ oc get clusterrolebinding spo-metrics-client -o wide
```

出力例

```
NAME                ROLE                            AGE   USERS   GROUPS   SERVICEACCOUNTS
spo-metrics-client   ClusterRole/spo-metrics-client   35m                    openshift-security-
profiles/default
```

7.8.6.1. コントローラーランタイムメトリクス

controller-runtime **metrics** と DaemonSet エンドポイント **metrics-spod** は、一連のデフォルトメトリクスを提供します。デーモンによって追加のメトリックが提供され、常に **security_profiles_operator_** という接頭辞が付けられます。

表7.1 利用可能なコントローラーランタイムメトリクス

メトリックキー	可能なラベル	型	目的
seccomp_profile_total	**operation={delete,update}**	カウンター	seccomp プロファイル操作の量。
seccomp_profile_audit_total	**node、namespace、pod、container、executable、syscall**	カウンター	seccomp プロファイル監査操作の量。ログエンリッチャーを有効にする必要があります。
seccomp_profile_bpf_total	**node、mount_namespace、profile**	カウンター	seccomp プロファイル bpf 操作の量。bpf レコーダを有効にする必要があります。
seccomp_profile_error_total	**reason={ SeccompNotSupportedOnNode, InvalidSeccompProfile, CannotSaveSeccompProfile, CannotRemoveSeccompProfile, CannotUpdateSeccompProfile, CannotUpdateNodeStatus }**	カウンター	seccomp プロファイルエラーの量。
selinux_profile_total	**operation={delete,update}**	カウンター	SELinux プロファイル操作の量。
selinux_profile_audit_total	**node、namespace、pod、container、executable、scontext、tcontext**	カウンター	SELinux プロファイルの監査操作の量。ログエンリッチャーを有効にする必要があります。

メトリックキー	可能なラベル	型	目的
selinux_profile_error_total	**reason={ CannotSaveSelinuxPolicy, CannotUpdatePolicyStatus, CannotRemoveSelinuxPolicy, CannotContactSelinuxd, CannotWritePolicyFile, CannotGetPolicyStatus }**	カウンター	SELinux プロファイルエラーの量。

7.8.7. ログエンリッチャーの使用

Security Profiles Operator にはログ強化機能が含まれており、デフォルトでは無効になっています。ログエンリッチャーコンテナーは、ローカルノードから監査ログを読み取るための **privileged** 権限で実行されます。ログエンリッチャーは、ホスト PID namespace **hostPID** 内で実行されます。

> **重要**
>
> ログエンリッチャーには、ホストプロセスを読み取る権限が必要です。

手順

1. 次のコマンドを実行して、**spod** 設定にパッチを適用し、ログエンリッチャーを有効にします。

```
$ oc -n openshift-security-profiles patch spod spod \
    --type=merge -p '{"spec":{"enableLogEnricher":true}}'
```

出力例

```
securityprofilesoperatordaemon.security-profiles-operator.x-k8s.io/spod patched
```

> **注記**
>
> Security Profiles Operator は、**spod** デーモンセットを自動的に再デプロイします。

2. 次のコマンドを実行して、監査ログを表示します。

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

出力例

```
I0623 12:51:04.257814 1854764 deleg.go:130] setup "msg"="starting component: log-
```

```
enricher"  "buildDate"="1980-01-01T00:00:00Z" "compiler"="gc" "gitCommit"="unknown"
"gitTreeState"="clean" "goVersion"="go1.16.2" "platform"="linux/amd64" "version"="0.4.0-
dev"
I0623 12:51:04.257890 1854764 enricher.go:44] log-enricher "msg"="Starting log-enricher on
node: 127.0.0.1"
I0623 12:51:04.257898 1854764 enricher.go:46] log-enricher "msg"="Connecting to local
GRPC server"
I0623 12:51:04.258061 1854764 enricher.go:69] log-enricher "msg"="Reading from file
/var/log/audit/audit.log"
2021/06/23 12:51:04 Seeked /var/log/audit/audit.log - &{Offset:0 Whence:2}
```

7.8.7.1. ログエンリッチャーを使用してアプリケーションをトレースする

Security Profiles Operator ログエンリッチャーを使用して、アプリケーションをトレースできます。

手順

1. アプリケーションをトレースするには、**SeccompProfile** ロギングプロファイルを作成します。

```
apiVersion: security-profiles-operator.x-k8s.io/v1beta1
kind: SeccompProfile
metadata:
  name: log
spec:
  defaultAction: SCMP_ACT_LOG
```

2. プロファイルを使用する Pod オブジェクトを作成します。

```
apiVersion: v1
kind: Pod
metadata:
  name: log-pod
  namespace: default
spec:
  securityContext:
    runAsNonRoot: true
    seccompProfile:
      type: Localhost
      localhostProfile: operator/log.json
  containers:
  - name: log-container
    image: quay.io/security-profiles-operator/test-nginx-unprivileged:1.21
    securityContext:
      allowPrivilegeEscalation: false
      capabilities:
        drop: [ALL]
```

3. 次のコマンドを実行して、ログエンリッチャーの出力を調べます。

```
$ oc -n openshift-security-profiles logs -f ds/spod log-enricher
```

例7.1 出力例

```
…
I0623 12:59:11.479869 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=3 "syscallName"="close"
"timestamp"="1624453150.205:1061" "type"="seccomp"
I0623 12:59:11.487323 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=157 "syscallName"="prctl"
"timestamp"="1624453150.205:1062" "type"="seccomp"
I0623 12:59:11.492157 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/" "namespace"="default" "node"="127.0.0.1"
"pid"=1905792 "pod"="log-pod" "syscallID"=157 "syscallName"="prctl"
"timestamp"="1624453150.205:1063" "type"="seccomp"
…
I0623 12:59:20.258523 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=12 "syscallName"="brk"
"timestamp"="1624453150.235:2873" "type"="seccomp"
I0623 12:59:20.263349 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=21
"syscallName"="access" "timestamp"="1624453150.235:2874" "type"="seccomp"
I0623 12:59:20.354091 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=257
"syscallName"="openat" "timestamp"="1624453150.235:2875" "type"="seccomp"
I0623 12:59:20.358844 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=5 "syscallName"="fstat"
"timestamp"="1624453150.235:2876" "type"="seccomp"
I0623 12:59:20.363510 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=9 "syscallName"="mmap"
"timestamp"="1624453150.235:2877" "type"="seccomp"
I0623 12:59:20.454127 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=3 "syscallName"="close"
"timestamp"="1624453150.235:2878" "type"="seccomp"
I0623 12:59:20.458654 1854764 enricher.go:111] log-enricher "msg"="audit"
"container"="log-container" "executable"="/usr/sbin/nginx" "namespace"="default"
"node"="127.0.0.1" "pid"=1905792 "pod"="log-pod" "syscallID"=257
"syscallName"="openat" "timestamp"="1624453150.235:2879" "type"="seccomp"
…
```

7.8.8. Webhook の設定

プロファイルバインディングおよびプロファイル記録オブジェクトは Webhook を使用できます。プロファイルバインディングおよび記録オブジェクトの設定は **MutatingWebhookConfiguration** CR であり、Security Profiles Operator によって管理されます。

Webhook 設定を変更するために、**spod** CR は、**failurePolicy**、**namespaceSelector**、および **objectSelector** 変数の変更を許可する **webhookOptions** フィールドを公開します。これにより、Webhook を "ソフトフェイル" に設定したり、namespace のサブセットに制限したりして、Webhook が失敗した場合でも他の namespace やリソースに影響を与えないようにすることができます。

手順

1. 次のパッチファイルを作成して、**recording.spo.io** Webhook 設定を設定し、 **spo-record=true** でラベル付けされた Pod のみを記録します。

```
spec:
  webhookOptions:
    - name: recording.spo.io
      objectSelector:
        matchExpressions:
          - key: spo-record
            operator: In
            values:
              - "true"
```

2. 次のコマンドを実行して、**spod/spod** インスタンスにパッチを適用します。

```
$ oc -n openshift-security-profiles patch spod \
    spod -p $(cat /tmp/spod-wh.patch) --type=merge
```

3. 結果の **MutatingWebhookConfiguration** オブジェクトを表示するには、次のコマンドを実行します。

```
$ oc get MutatingWebhookConfiguration \
    spo-mutating-webhook-configuration -oyaml
```

7.9. SECURITY PROFILES OPERATOR のトラブルシューティング

Security Profiles Operator をトラブルシューティングして、問題を診断するか、バグレポートで情報を提供します。

7.9.1. seccomp プロファイルの検査

seccomp プロファイルが破損していると、ワークロードが中断される可能性があります。他のワークロードがパス **/var/lib/kubelet/seccomp/operator** のどの部分もマップできないようにすることで、ユーザーがシステムを悪用できないようにします。

手順

1. 次のコマンドを実行して、プロファイルが調整されていることを確認します。

```
$ oc -n openshift-security-profiles logs openshift-security-profiles-<id>
```

例7.2 出力例

```
I1019 19:34:14.942464       1 main.go:90] setup "msg"="starting openshift-security-
profiles"  "buildDate"="2020-10-19T19:31:24Z" "compiler"="gc"
"gitCommit"="a3ef0e1ea6405092268c18f240b62015c247dd9d" "gitTreeState"="dirty"
"goVersion"="go1.15.1" "platform"="linux/amd64" "version"="0.2.0-dev"
I1019 19:34:15.348389       1 listener.go:44] controller-runtime/metrics "msg"="metrics
server is starting to listen"  "addr"=":8080"
I1019 19:34:15.349076       1 main.go:126] setup "msg"="starting manager"
I1019 19:34:15.349449       1 internal.go:391] controller-runtime/manager "msg"="starting
```

```
metrics server"  "path"="/metrics"
I1019 19:34:15.350201       1 controller.go:142] controller "msg"="Starting EventSource"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile" "source"={"Type":{"metadata":
{"creationTimestamp":null},"spec":{"defaultAction":""}}}
I1019 19:34:15.450674       1 controller.go:149] controller "msg"="Starting Controller"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile"
I1019 19:34:15.450757       1 controller.go:176] controller "msg"="Starting workers"
"controller"="profile" "reconcilerGroup"="security-profiles-operator.x-k8s.io"
"reconcilerKind"="SeccompProfile" "worker count"=1
I1019 19:34:15.453102       1 profile.go:148] profile "msg"="Reconciled profile from
SeccompProfile" "namespace"="openshift-security-profiles" "profile"="nginx-1.19.1"
"name"="nginx-1.19.1" "resource version"="728"
I1019 19:34:15.453618       1 profile.go:148] profile "msg"="Reconciled profile from
SeccompProfile" "namespace"="openshift-security-profiles" "profile"="openshift-security-
profiles" "name"="openshift-security-profiles" "resource version"="729"
```

2. 次のコマンドを実行して、**seccomp** プロファイルが正しいパスに保存されていることを確認します。

```
$ oc exec -t -n openshift-security-profiles openshift-security-profiles-<id> \
    -- ls /var/lib/kubelet/seccomp/operator/my-namespace/my-workload
```

出力例

```
profile-block.json
profile-complain.json
```

7.10. SECURITY PROFILES OPERATOR のアンインストール

OpenShift Container Platform Web コンソールを使用してクラスターから Security Profiles Operator を削除できます。

7.10.1. Web コンソールを使用して Security Profiles Operator をアンインストールします

Security Profiles Operator を削除するには、最初に **seccomp** および SELinux プロファイルを削除する必要があります。プロファイルが削除されたら、**openshift-security-profiles** プロジェクトを削除することで、Operator とその namespace を削除できます。

前提条件

- **cluster-admin** パーミッションを持つアカウントを使用して OpenShift Container Platform クラスターにアクセスできる。
- Security Profiles Operator がインストールされます。

手順

OpenShift Container Platform Web コンソールを使用して Security Profiles Operator を削除するには、以下を実行します。

1. **Operators → Installed Operators** ページに移動します。
2. すべての **seccomp** プロファイル、SELinux プロファイル、および Webhook 設定を削除します。
3. **Administration → Operators → Installed Operators** ページに切り替えます。
4. **Security Profiles Operator** エントリーでオプションメニュー をクリックして、**Uninstall Operator** を選択します。
5. **Home → Projects** ページに切り替えます。
6. **security profiles** を検索します。
7. **openshift-security-profiles** プロジェクトの横にあるオプションメニュー をクリックし、**Delete Project** を選択します。
   a. ダイアログボックスに **openshift-security-profiles** と入力して削除を確認し、**Delete** をクリックします。
8. 次のコマンドを実行して **MutatingWebhookConfiguration** オブジェクトを削除します。

```
$ oc delete MutatingWebhookConfiguration spo-mutating-webhook-configuration
```

第8章 NBDE TANG SERVER OPERATOR

8.1. NBDE TANG SERVER OPERATOR の概要

Network-Bound Disk Encryption (NBDE) は、1 つ以上の専用ネットワークバインディングサーバーを使用して、LUKS 暗号化ボリュームの自動ロック解除を提供します。NBDE のクライアント側は Clevis 復号化ポリシーフレームワークと呼ばれ、サーバー側は Tang で表されます。

NBDE Tang Server Operator を使用すると、OpenShift Container Platform (OCP) 環境で 1 つ以上の Tang サーバーのデプロイメントを自動化できます。

8.2. NBDE TANG SERVER OPERATOR リリースノート

以下のリリースノートは、OpenShift Container Platform の NBDE Tang Server Operator の開発履歴を記録したものです。

RHEA-2023:7491

NBDE Tang Server Operator 1.0 が Red Hat OpenShift Enterprise 4 カタログでリリースされました。

RHEA-2024:0854

NBDE Tang Server Operator 1.0.1 が **alpha** チャネルから **stable** チャネルに移動されました。

RHBA-2024:8681

1.0.2 更新には、NBDE Tang Server Operator を使用してデプロイされたコンテナーの Container Health Index を最高グレードに上げる修正が含まれています。

RHEA-2024:10970

1.0.3 更新には、Container Health Index を再び最高グレードに上げる変更が含まれています。

RHBA-2025:0663

NBDE Tang Server Operator 1.1 では、**golang** パッケージがバージョン 1.23.2 で提供され、**golang.org/x/net/html** パッケージがバージョン 0.33.0 に更新されました。更新により、Container Health Index が増加します。

8.3. NBDE TANG SERVER OPERATOR について

NBDE Tang Server Operator を使用すると、自動化を実現するために OpenShift Container Platform が提供するツールを活用して、内部で Network Bound Disk Encryption (NBDE) を必要とする OpenShift Container Platform クラスター内の Tang サーバーのデプロイメントを自動化できます。

NBDE Tang Server Operator はインストールプロセスを簡略化し、マルチレプリカデプロイメント、スケーリング、トラフィック負荷分散などの、OpenShift Container Platform 環境が提供するネイティブ機能を使用します。Operator は、手動で実行するとエラーが発生しやすい特定の操作も自動化できます。以下はその例です。

- サーバーのデプロイと設定
- 鍵のローテーション
- 隠し鍵の削除

NBDE Tang Server Operator は Operator SDK を使用して実装されます。これを使用すると、カスタムリソース定義 (CRD) を通じて OpenShift に 1 つ以上の Tang サーバーをデプロイできます。

8.3.1. 関連情報

- Red Hat Hybrid Cloud ブログ記事 Tang-Operator: Providing NBDE in OpenShift
- tang-operator Github プロジェクト
- RHEL 9 セキュリティーハードニングドキュメントの ポリシーベースの復号化を使用した暗号化ボリュームの自動ロック解除の設定

8.4. NBDE TANG SERVER OPERATOR のインストール

NBDE Tang Operator は、Web コンソールを使用するか、CLI から **oc** コマンドを使用してインストールできます。

8.4.1. Web コンソールを使用して NBDE Tang Server Operator をインストールする

Web コンソールを使用して、OperatorHub から NBDE Tang Server Operator をインストールできます。

前提条件

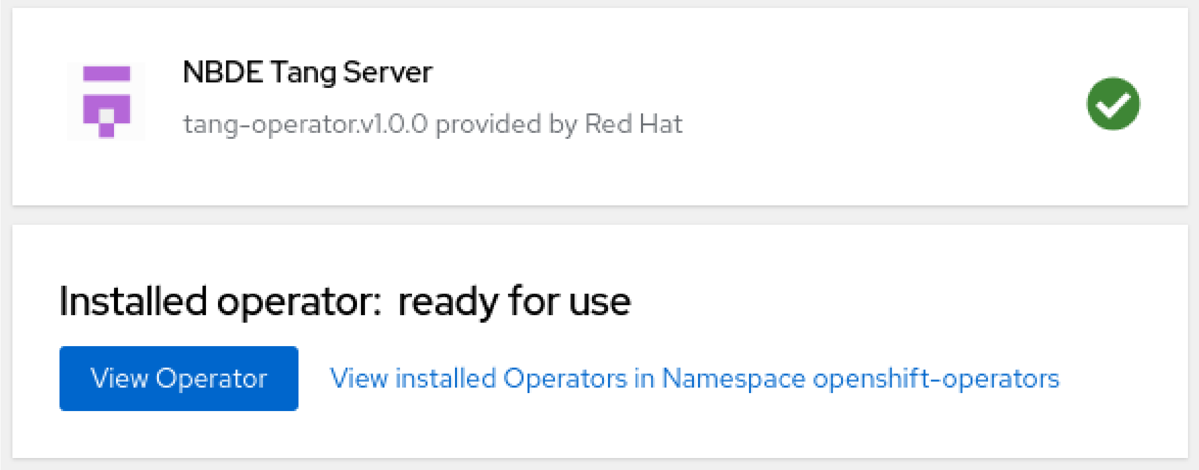
- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. NBDE Tang Server Operator を検索します。

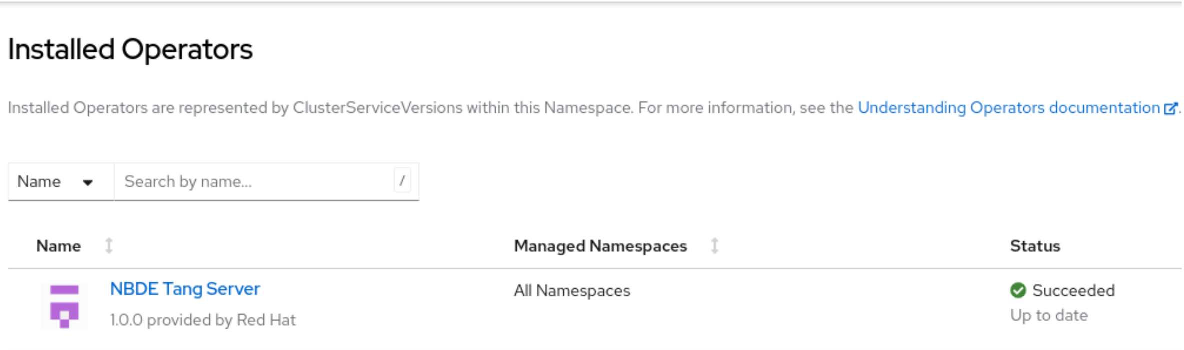
3. **Install** をクリックします。
4. **Operator Installation** 画面で、**Update channel**、**Version**、**Installation mode**、**Installed Namespace**、および **Update approval** フィールドは、デフォルト値のままにします。

5. **Install** をクリックしてインストールオプションを確認すると、コンソールにインストールの確認が表示されます。



検証

1. **Operators → Installed Operators** ページに移動します。
2. NBDE Tang Server Operator がインストールされており、そのステータスが **Succeeded** であることを確認します。



8.4.2. CLI を使用して NBDE Tang Server Operator をインストールする

CLI を使用して、OperatorHub から NBDE Tang Server Operator をインストールできます。

前提条件

- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. 次のコマンドを使用して、OperatorHub で利用可能なオペレータをリスト表示し、出力を Tang 関連の結果に制限します。

```
$ oc get packagemanifests -n openshift-marketplace | grep tang
```

出力例

```
tang-operator        Red Hat
```

この場合、対応するパッケージマニフェスト名は **tang-operator** です。

2. **Subscription** オブジェクト YAML ファイルを作成して、namespace を NBDE Tang Server Operator をサブスクライブします (例: **tang-operator.yaml**)。

tang-operator のサブスクリプション YAML の例

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: tang-operator
  namespace: openshift-operators
spec:
  channel: stable (1)
  installPlanApproval: Automatic
  name: tang-operator (2)
  source: redhat-operators (3)
  sourceNamespace: openshift-marketplace (4)
```

(1) Operator のサブスクライブ元のチャネル名を指定します。

(2) サブスクライブする Operator の名前を指定します。

(3) Operator を提供する CatalogSource の名前を指定します。

(4) CatalogSource の namespace。デフォルトの OperatorHub CatalogSource には **openshift-marketplace** を使用します。

3. **Subscription** をクラスターに適用します。

```
$ oc apply -f tang-operator.yaml
```

検証

- NBDE Tang Server Operator コントローラーが **openshift-operators** namespace で実行されていることを確認します。

```
$ oc -n openshift-operators get pods
```

出力例

```
NAME                                               READY   STATUS    RESTARTS   AGE
tang-operator-controller-manager-694b754bd6-4zk7x   2/2     Running   0          12s
```

8.5. NBDE TANG SERVER OPERATOR を使用した TANG サーバーの設定と管理

NBDE Tang Server Operator を使用すると、Tang サーバーをデプロイして迅速に設定できます。デプロイされた Tang サーバーでは、既存のキーをリストし、それらをローテーションできます。

8.5.1. NBDE Tang Server Operator を使用して Tang サーバーをデプロイする

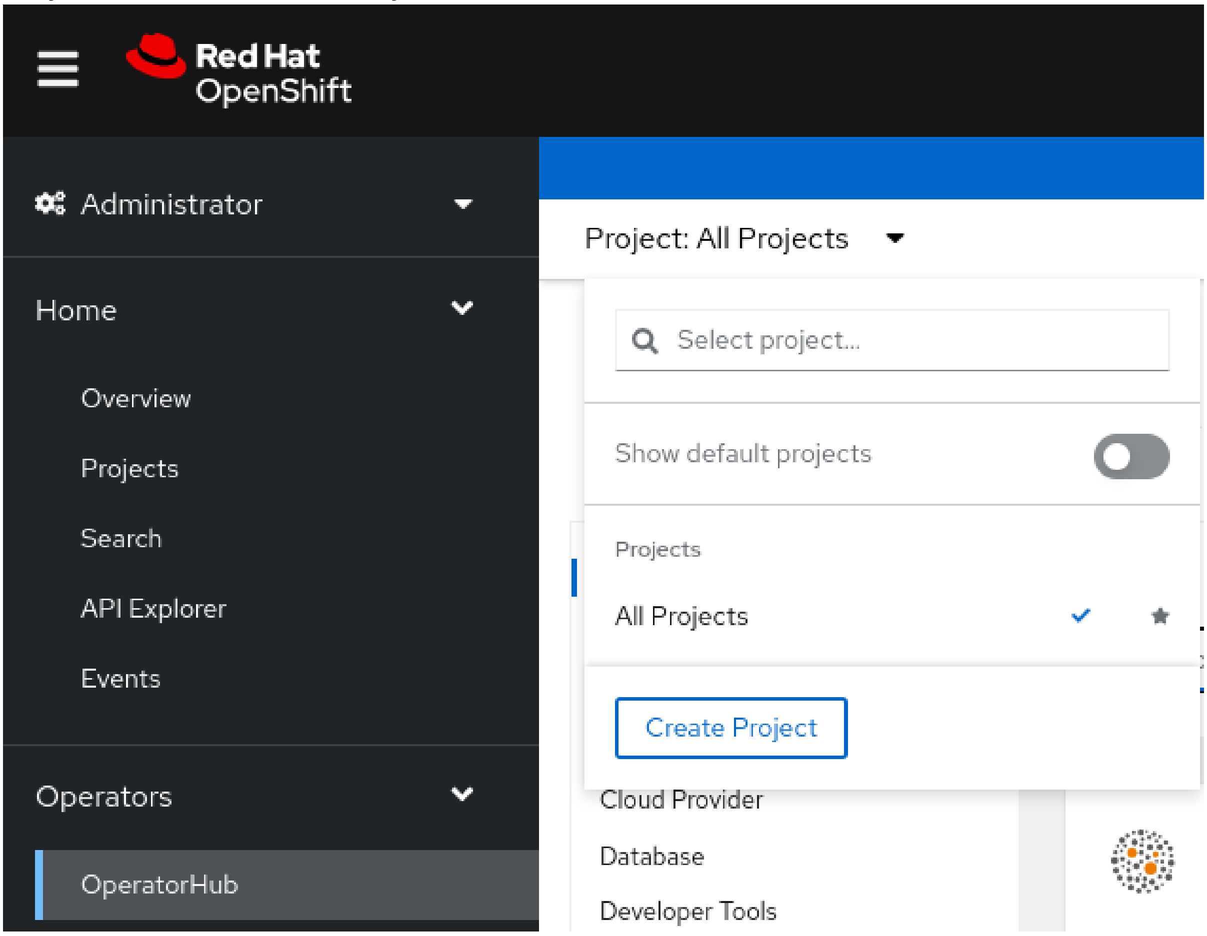
Web コンソールで NBDE Tang Server Operator を使用すると、1 つ以上の Tang サーバーをデプロイして迅速に設定できます。

前提条件

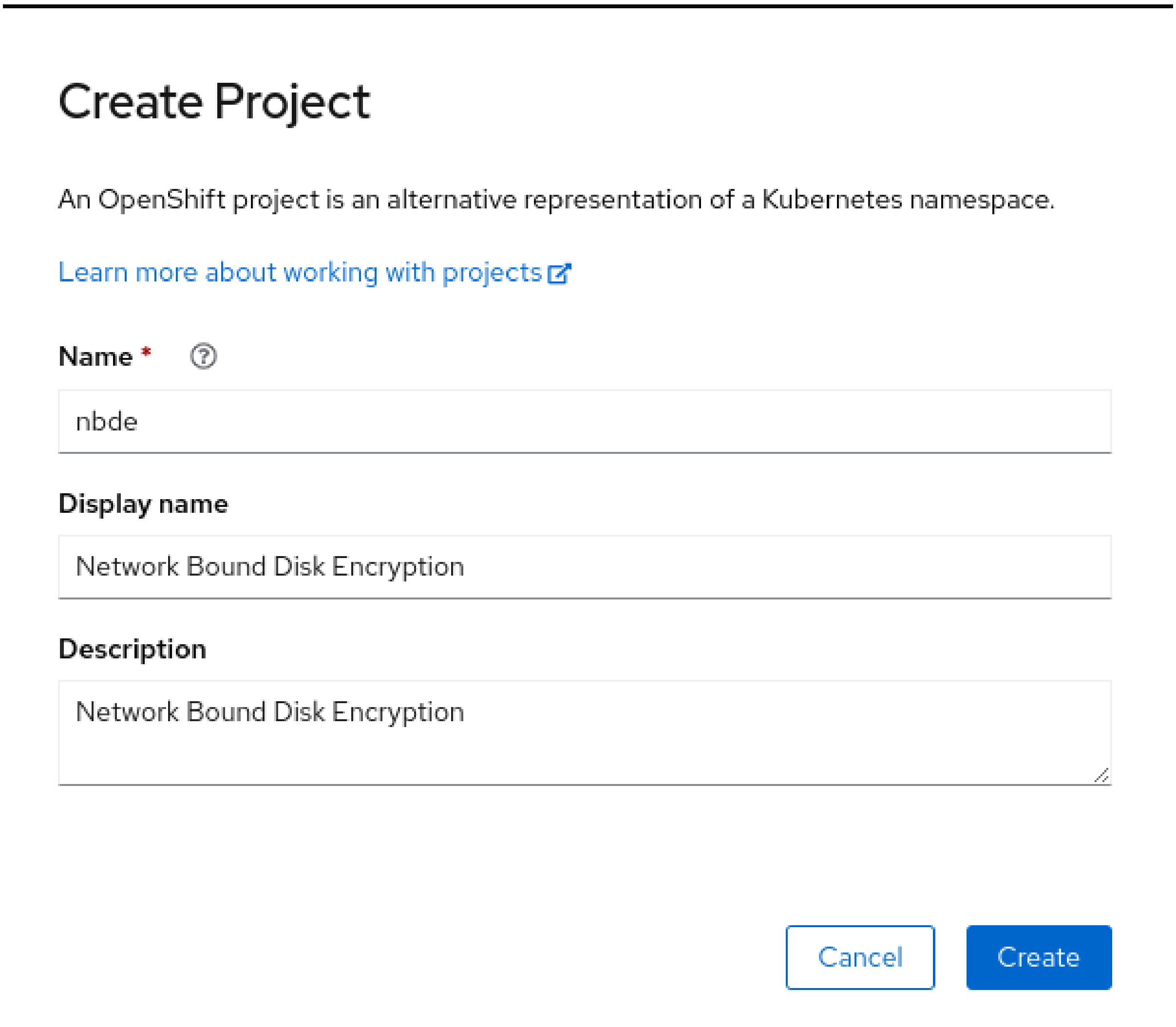
- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- OCP クラスターに NBDE Tang Server Operator がインストールされている。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → OperatorHub** ページに移動します。
2. **Project** を選択し、**Create Project** をクリックします。

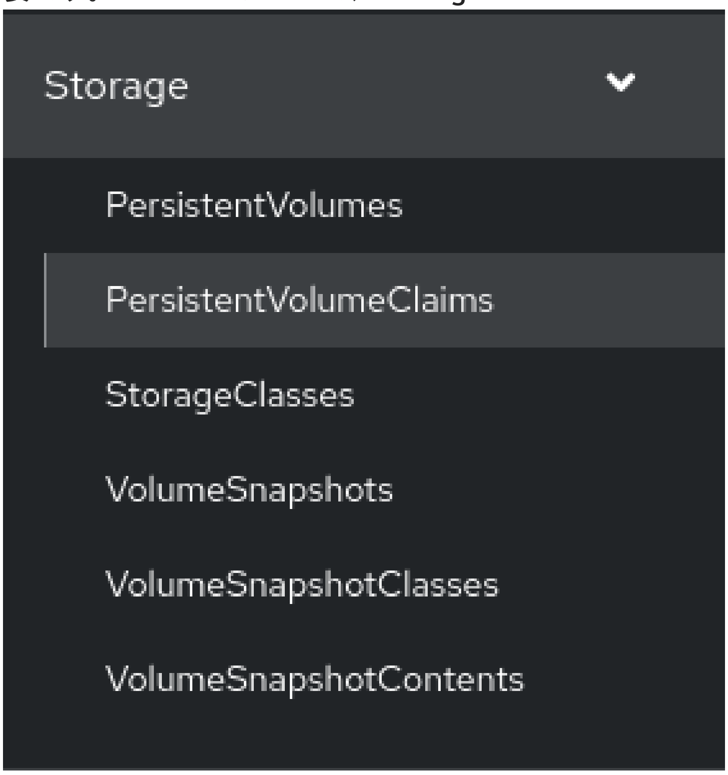


3. **Create Project** ページで、次のような必須情報を入力します。



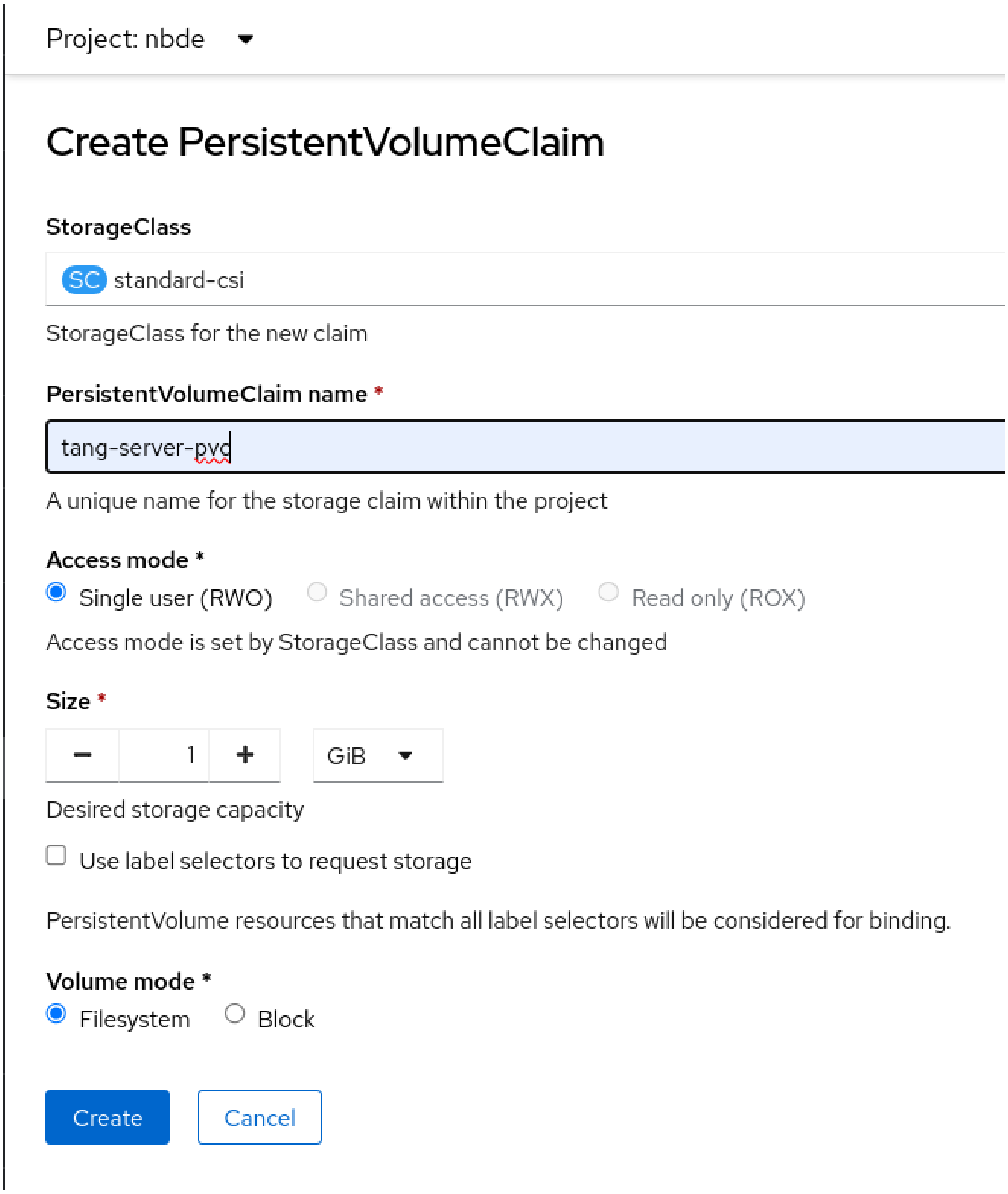
4. **Create** をクリックします。

5. NBDE Tang Server レプリカには、暗号鍵を保存するためにの永続ボリューム要求 (PVC) が必要です。Web コンソールで、**Storage** → **PersistentVolumeClaims** に移動します。



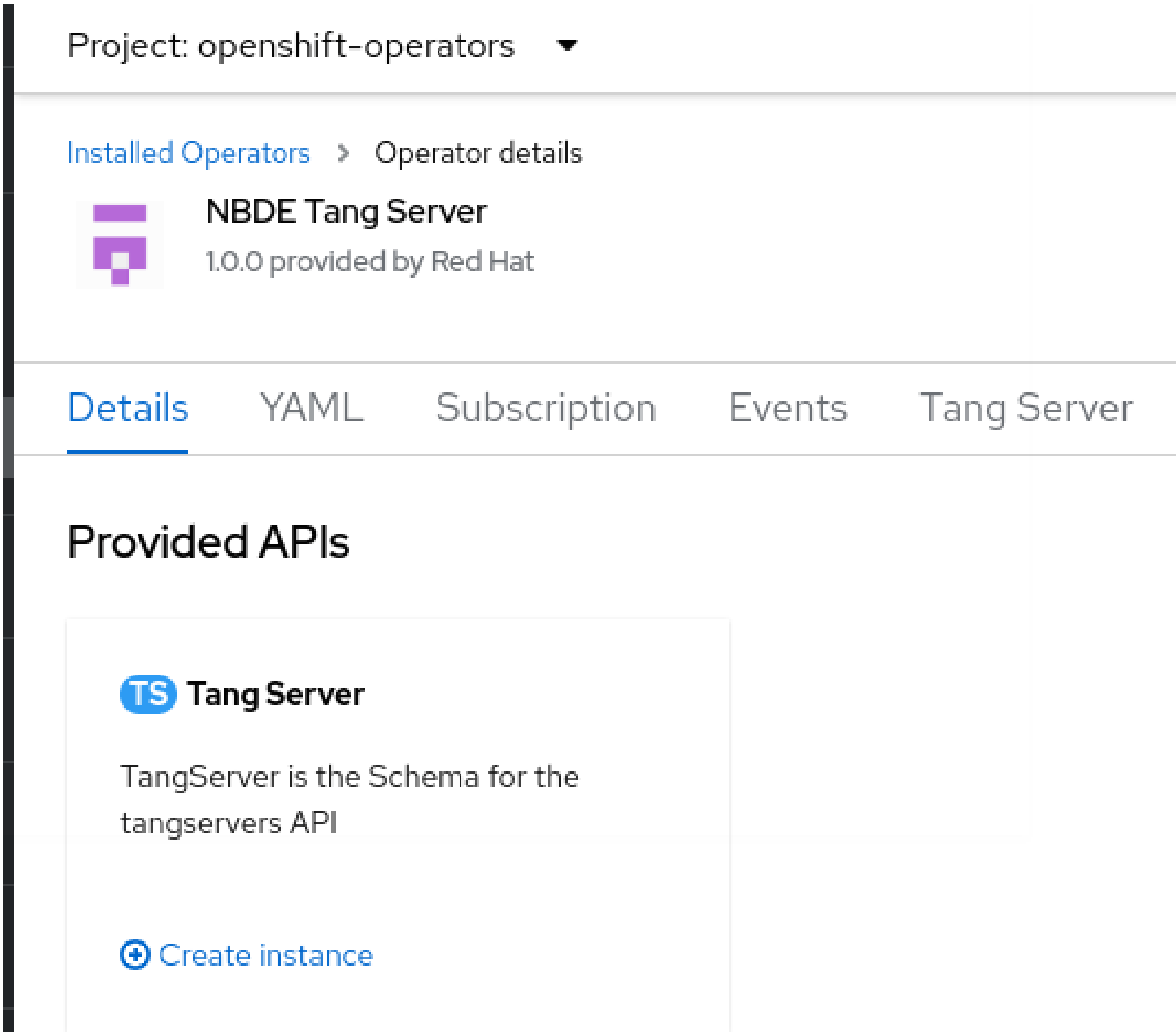
6. 次の **PersistentVolumeClaims** 画面で、**Create PersistentVolumeClaim** をクリックします。

7. **Create PersistentVolumeClaim** ページで、デプロイメントシナリオに適したストレージを選択します。暗号鍵をローテーションする頻度を検討してください。PVC に名前を付け、要求されたストレージ容量を選択します。以下はその例です。

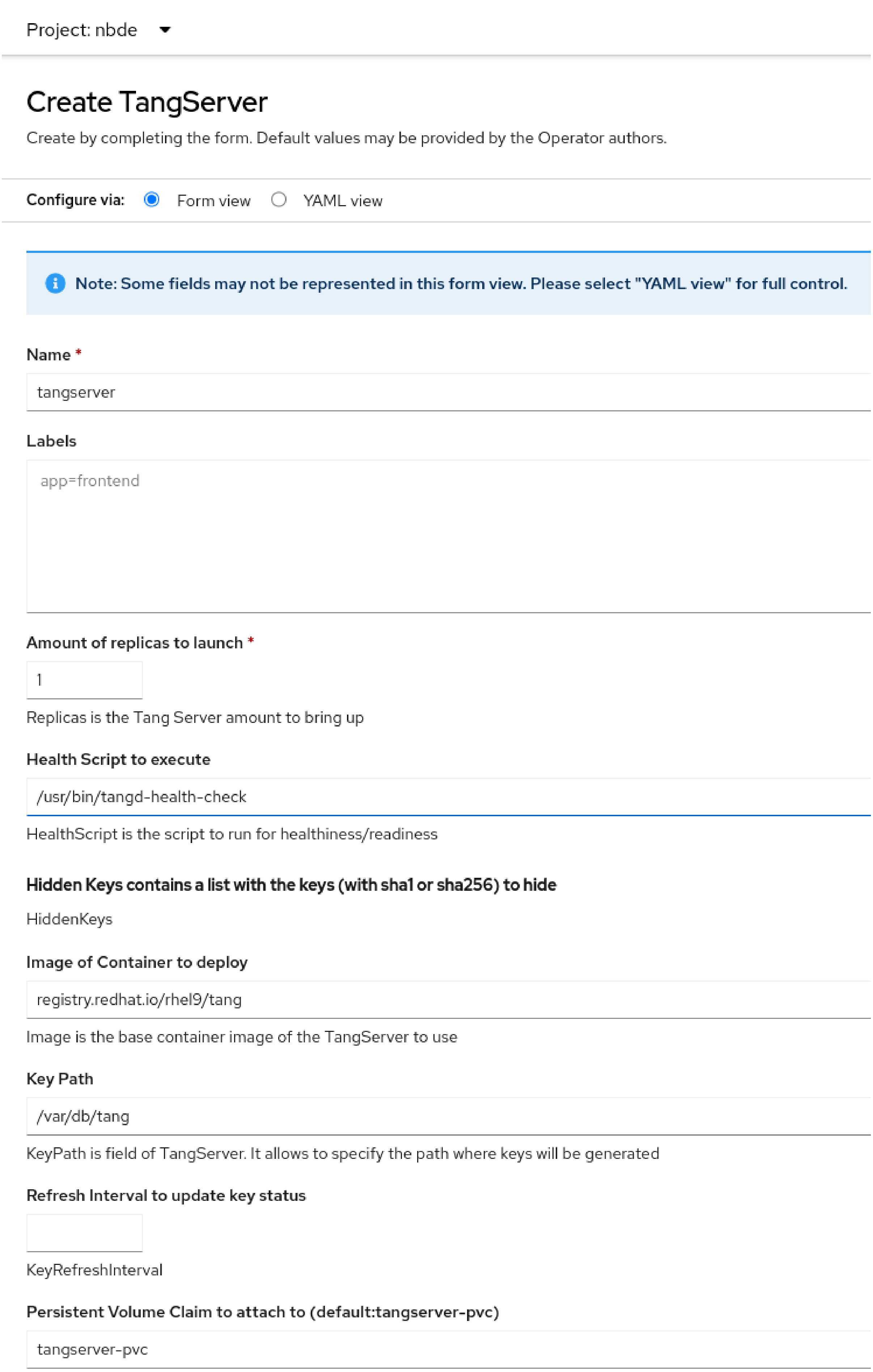


8. **Operators → Installed Operators** に移動し、**NBDE Tang Server** をクリックします。

9. **Create instance** をクリックします。



10. **Create TangServer** ページで、Tang Server インスタンスの名前とレプリカの数を選択し、前に作成した永続ボリューム要求の名前を指定します。以下はその例です。



11. 必要な値を入力し、シナリオのデフォルト値とは異なる設定を変更してから **Create** をクリックします。

8.5.2. NBDE Tang Server Operator を使用して鍵をローテートする

NBDE Tang Server Operator を使用すると、Tang サーバーの鍵をローテートすることも可能です。鍵をローテートするのに適した間隔は、アプリケーション、鍵のサイズ、および組織のポリシーにより異なります。

前提条件

- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- NBDE Tang Server Operator を使用して Tang サーバーを OpenShift クラスターにデプロイしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Tang サーバー上の既存の鍵をリストします。以下はその例です。

```
$ oc -n nbde describe tangserver
```

出力例

```
…
Status:
  Active Keys:
 File Name:   QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
 Generated:   2022-02-08 15:44:17.030090484 +0000
 sha1:        PvYQKtrTuYsMV2AomUeHrUWkCGg
 sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. アクティブな鍵を非表示の鍵に移動ための YAML ファイル (例: **minimal-keyretrieve-rotate-tangserver.yaml**) を作成します。

tang-operator の鍵ローテーション YAML の例

```
apiVersion: daemons.redhat.com/v1alpha1
kind: TangServer
metadata:
  name: tangserver
  namespace: nbde
  finalizers:
    - finalizer.daemons.tangserver.redhat.com
spec:
  replicas: 1
  hiddenKeys:
    - sha1: "PvYQKtrTuYsMV2AomUeHrUWkCGg" (1)
```

(1) アクティブな鍵の SHA-1 サムプリントを指定して、鍵をローテーションします。

3. YAML ファイルを適用します。

```
$ oc apply -f minimal-keyretrieve-rotate-tangserver.yaml
```

検証

1. 設定に応じて一定の時間が経過した後、以前の **activeKey** 値が新しい **hiddenKey** 値になり、**activeKey** キーファイルが新しく生成されていることを確認します。以下はその例です。

   ```
   $ oc -n nbde describe tangserver
   ```

   出力例

   ```
   …
   Spec:
     Hidden Keys:
       sha1:    PvYQKtrTuYsMV2AomUeHrUWkCGg
     Replicas:  1
   Status:
     Active Keys:
       File Name:  T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY.jwk
       Generated:  2023-10-25 15:38:18.134939752 +0000
       sha1:       vVxkNCNq7gygeeA9zrHrbc3_NZ4
       sha256:     T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY
     Hidden Keys:
       File Name:          .QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg.jwk
       Generated:          2023-10-25 15:37:29.126928965 +0000
       Hidden:             2023-10-25 15:38:13.515467436 +0000
       sha1:               PvYQKtrTuYsMV2AomUeHrUWkCGg
       sha256:             QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
   …
   ```

8.5.3. NBDE Tang Server Operator を使用して非表示の鍵を削除する

Tang サーバー鍵をローテートすると、それまでアクティブだった鍵は非表示になり、Tang インスタンスによってアドバタイズされなくなります。NBDE Tang Server Operator を使用して、使用されなくなった暗号鍵を削除できます。

WARNING

バインドされたすべての Clevis クライアントが新しい鍵を使用していることが確認できない限り、非表示の鍵を削除しないでください。

前提条件

- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- NBDE Tang Server Operator を使用して Tang サーバーを OpenShift クラスターにデプロイしている。
- OpenShift CLI (**oc**) がインストールされている。

手順

1. Tang サーバー上の既存の鍵をリストします。以下はその例です。

```
$ oc -n nbde describe tangserver
```

出力例

```
…
Status:
  Active Keys:
   File Name:   PvYQKtrTuYsMV2AomUeHrUWkCGg.jwk
   Generated:   2022-02-08 15:44:17.030090484 +0000
   sha1:      PvYQKtrTuYsMV2AomUeHrUWkCGg
   sha256:      QS82aXnPKA4XpfHr3umbA0r2iTbRcpWQ0VI2Qdhi6xg
…
```

2. 非表示の鍵をすべて削除するための YAML ファイル (例: **hidden-keys-deletion-tangserver.yaml**) を作成します。

tang-operator の hidden-keys-deletion YAML の例

```
apiVersion: daemons.redhat.com/v1alpha1
kind: TangServer
metadata:
  name: tangserver
  namespace: nbde
  finalizers:
    - finalizer.daemons.tangserver.redhat.com
spec:
  replicas: 1
  hiddenKeys: [] 1
```

1 **hiddenKeys** エントリーの値が空の配列である場合、Tang サーバーに非表示の鍵が保存されていないことを示します。

3. YAML ファイルを適用します。

```
$ oc apply -f hidden-keys-deletion-tangserver.yaml
```

検証

1. 設定に応じて一定の時間が経過した後、以前のアクティブな鍵が引き続き存在し、使用可能な非表示の鍵もないことを確認します。以下はその例です。

```
$ oc -n nbde describe tangserver
```

出力例

```
…
Spec:
  Hidden Keys:
    sha1:   PvYQKtrTuYsMV2AomUeHrUWkCGg
  Replicas:  1
Status:
  Active Keys:
```

```
    File Name:  T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY.jwk
    Generated:  2023-10-25 15:38:18.134939752 +0000
    sha1:       vVxkNCNq7gygeeA9zrHrbc3_NZ4
    sha256:     T-0wx1HusMeWx4WMOk4eK97Q5u4dY5tamdDs7_ughnY
Status:
  Ready:              1
  Running:            1
  Service External URL:  http://35.222.247.84:7500/adv
  Tang Server Error:     No
Events:
…
```

8.6. NBDE TANG SERVER OPERATOR を使用してデプロイされた TANG サーバーの URL の特定

Tang サーバーによってアドバタイズされた暗号鍵を使用するように Clevis クライアントを設定する前に、サーバーの URL を特定する必要があります。

8.6.1. Web コンソールを使用して NBDE Tang Server Operator の URL を特定する

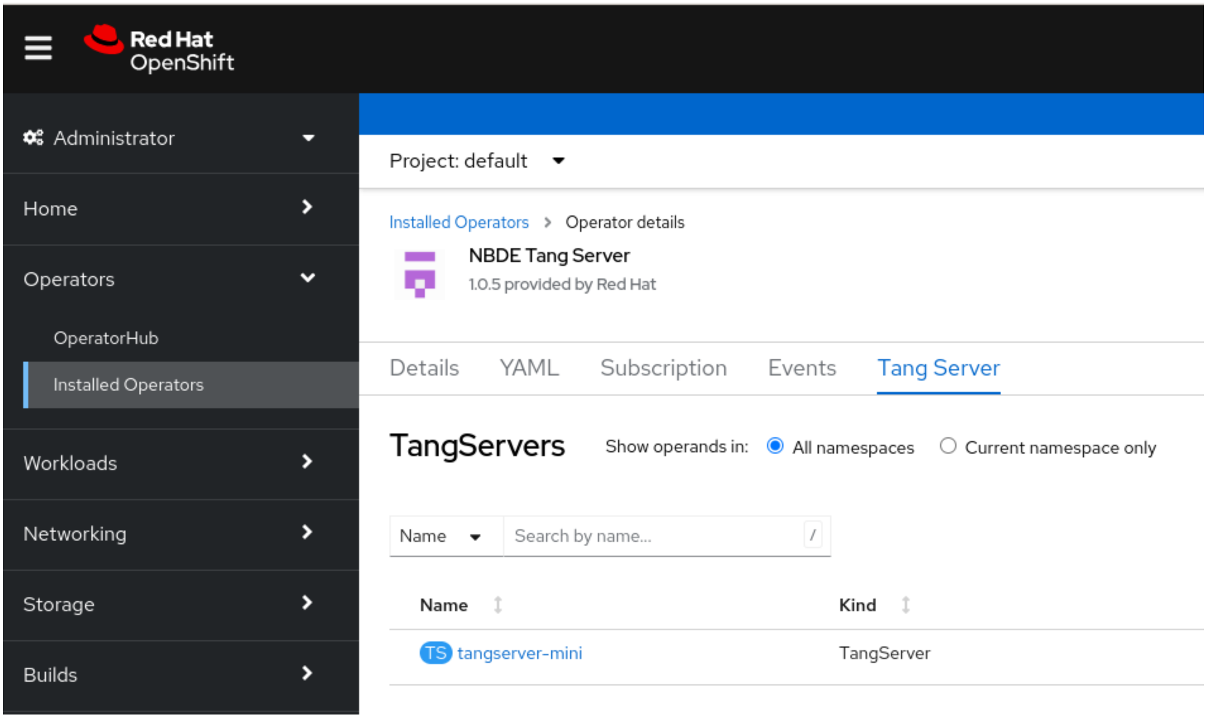
OpenShift Container Platform の Web コンソールを使用して、OperatorHub から NBDE Tang Server Operator でデプロイされた Tang サーバーの URL を特定できます。URL を特定した後、Tang サーバーによってアドバタイズされた鍵を使用して、自動的にロック解除する LUKS 暗号化ボリュームが含まれるクライアントで **clevis luks bind** コマンドを使用します。Clevis を使用してクライアントの設定を記述する手順の詳細は、RHEL 9 セキュリティーハードニングドキュメントの LUKS 暗号化ボリュームの手動登録を設定する セクションを参照してください。

前提条件

- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- OpenShift クラスター上で、NBDE Tang Server Operator を使用して Tang サーバーをデプロイしている。

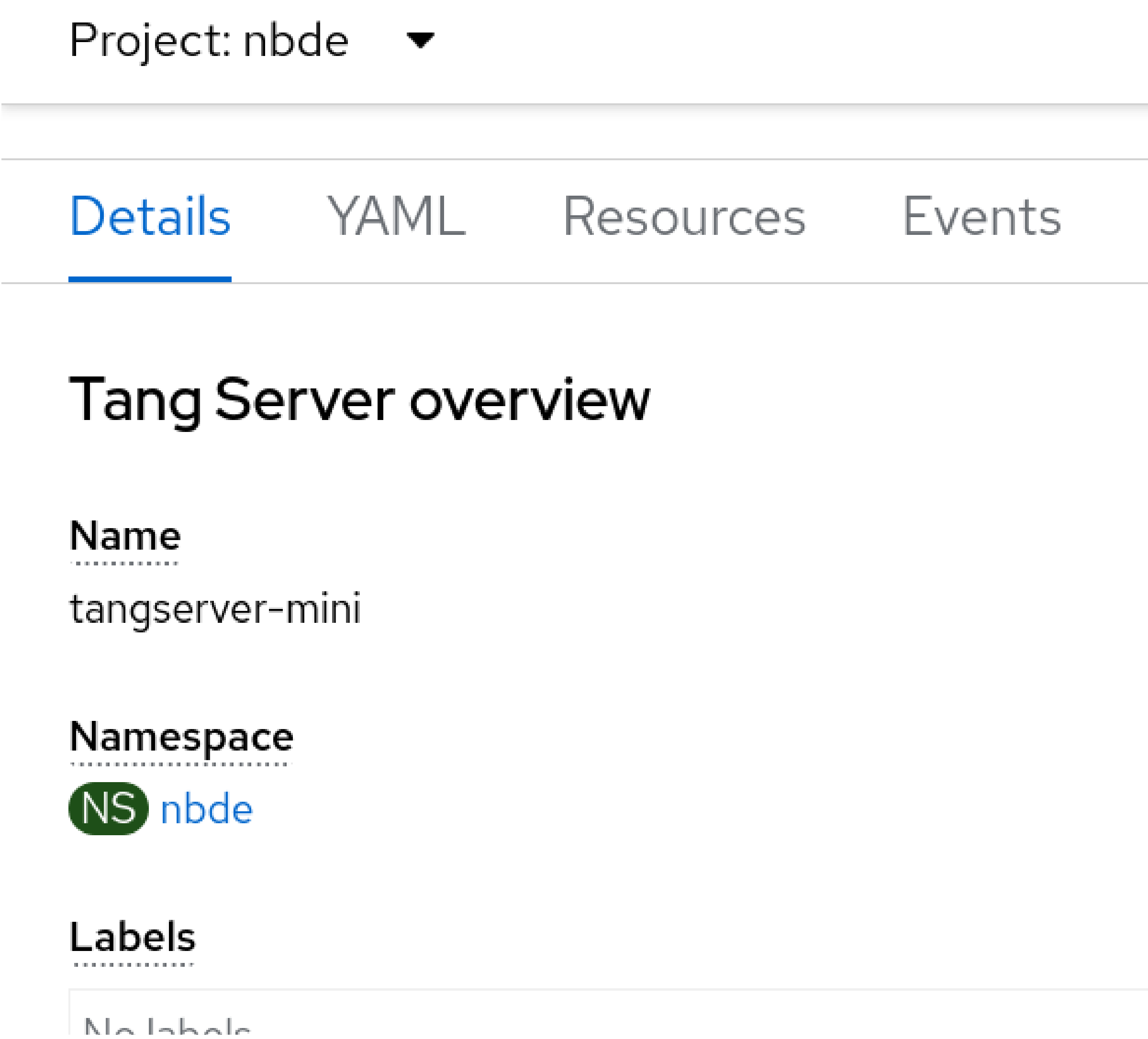
手順

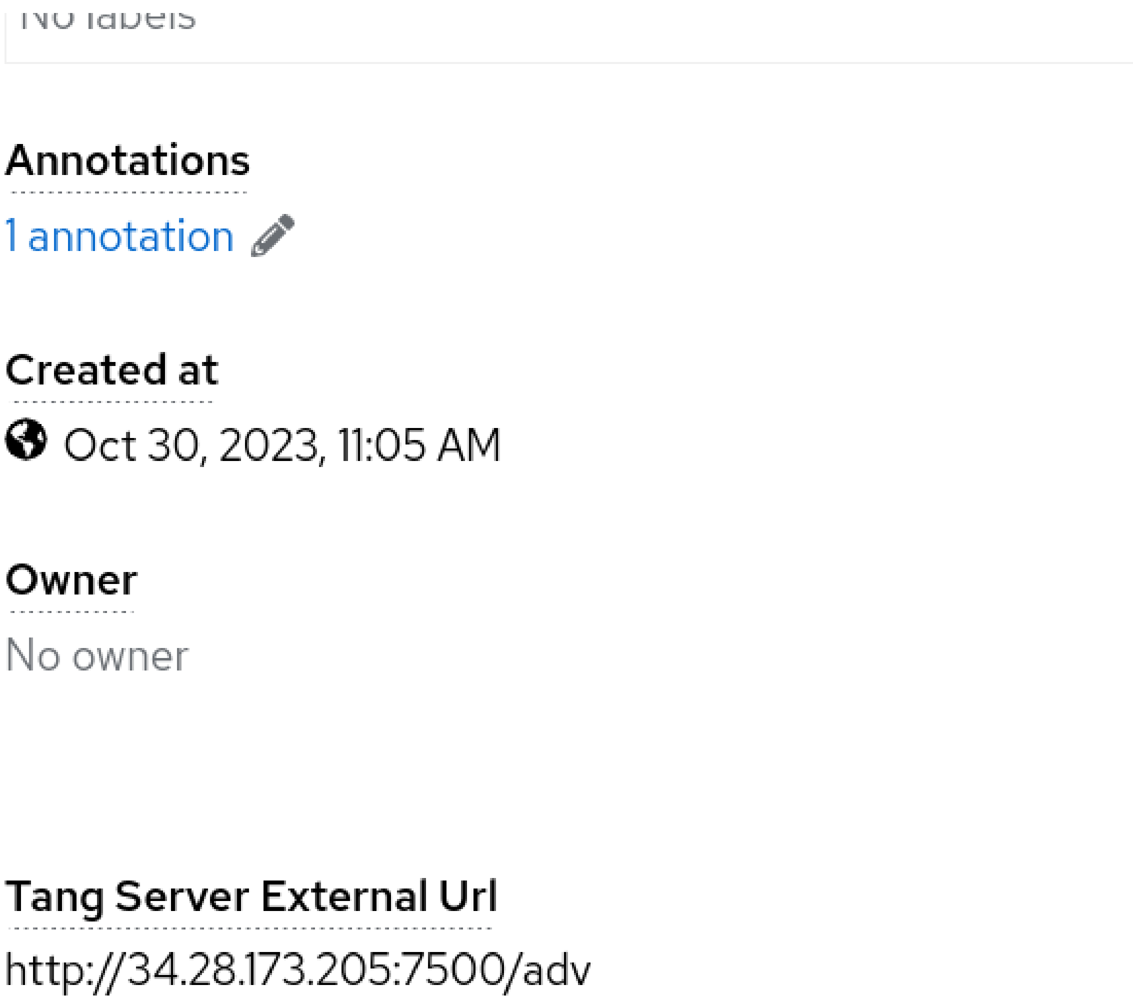
1. OpenShift Container Platform の Web コンソールで、**Operators → Installed Operators → Tang Server** に移動します。
2. NBDE Tang Server Operator の詳細ページで、**Tang Server** を選択します。



3. デプロイされ、クラスターで使用できる Tang サーバーのリストが表示されます。Clevis クライアントとバインドする Tang サーバーの名前をクリックします。

4. Web コンソールには、選択した Tang サーバーの概要が表示されます。Tang サーバーの URL は、画面の **Tang Server External Url** セクションで確認できます。





この例では、Tang サーバーの URL は **http://34.28.173.205:7500** です。

検証

- Tang サーバーがアドバタイズしていることを確認するには、**curl**、**wget**、または同様のツールを使用します。以下はその例です。

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

出力例

```
{
  "payload": "eyJrZXIzIj...eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU...aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.2. CLI を使用して NBDE Tang Server Operator の URL を特定する

CLI を使用して、OperatorHub から NBDE Tang Server Operator でデプロイされた Tang サーバーの URL を特定できます。URL を特定した後、Tang サーバーによってアドバタイズされた鍵を使用して、自動的にロック解除する LUKS 暗号化ボリュームが含まれるクライアントで **clevis luks bind** コマンド

を使用します。Clevis を使用してクライアントの設定を記述する手順の詳細は、RHEL 9 セキュリティーハードニングドキュメントの LUKS 暗号化ボリュームの手動登録を設定する セクションを参照してください。

前提条件

- OpenShift Container Platform クラスターに対する **cluster-admin** 権限を持っている。
- OpenShift CLI (**oc**) がインストールされている。
- OpenShift クラスター上で、NBDE Tang Server Operator を使用して Tang サーバーをデプロイしている。

手順

1. Tang サーバーに関する詳細をリストします。以下はその例です。

```
$ oc -n nbde describe tangserver
```

出力例

```
…
Spec:
…
Status:
  Ready:                 1
  Running:               1
  Service External URL:  http://34.28.173.205:7500/adv
  Tang Server Error:     No
Events:
…
```

2. **/adv** 部分を除いた **Service External URL:** 項目の値を使用します。この例では、Tang サーバーの URL は **http://34.28.173.205:7500** です。

検証

- Tang サーバーがアドバタイズしていることを確認するには、**curl**、**wget**、または同様のツールを使用します。以下はその例です。

```
$ curl 2> /dev/null http://34.28.173.205:7500/adv  | jq
```

出力例

```
{
  "payload": "eyJrZXlzIj...eSJdfV19",
  "protected": "eyJhbGciOiJFUzUxMiIsImN0eSI6Imp3ay1zZXQranNvbiJ9",
  "signature": "AUB0qSFx0FJLeTU...aV_GYWlDx50vCXKNyMMCRx"
}
```

8.6.3. 関連情報

- RHEL 9 セキュリティーハードニングドキュメントの LUKS 暗号化ボリュームの手動登録の設定 セクション。

第9章 CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT

9.1. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT の概要

cert-manager Operator for Red Hat OpenShift は、アプリケーション証明書のライフサイクル管理を提供するクラスター全体のサービスです。cert-manager Operator for Red Hat OpenShift を使用すると、外部の証明機関と統合し、証明書のプロビジョニング、更新、および廃止を行うことができます。

9.1.1. cert-manager Operator for Red Hat OpenShift について

cert-manager プロジェクトは、認証局と証明書を Kubernetes API のリソースタイプとして導入します。これにより、クラスター内で作業する開発者にオンデマンドで証明書を提供できます。cert-manager Operator for Red Hat OpenShift は、cert-manager を OpenShift Container Platform クラスターに統合するためのサポートされた方法を提供します。

cert-manager Operator for Red Hat OpenShift は、以下の機能を提供します。

- 外部認証局との統合のサポート
- 証明書を管理するためのツール
- 開発者が証明書をセルフサービスする機能
- 証明書の更新が自動で行われています。

重要

クラスター内で、Red Hat OpenShift for OpenShift Container Platform の cert-manager Operator とコミュニティーの cert-manager Operator の両方を同時に使用しないでください。

また、単一の OpenShift クラスター内の複数の namespace に Red Hat OpenShift for OpenShift Container Platform の cert-manager Operator をインストールしないでください。

9.1.2. cert-manager Operator for Red Hat OpenShift 発行者プロバイダー

cert-manager Operator for Red Hat OpenShift は、次の発行者タイプでテストされています。

- 自動証明書管理環境 (ACME)
- 認証局 (CA)
- 自己署名
- Vault
- Venafi
- Nokia NetGuard Certificate Manager (NCM)
- Google cloud Certificate Authority Service (Google CAS)

9.1.2.1. 発行者タイプのテスト

次の表では、テストした各発行者タイプに対するテスト範囲を示しています。

発行者タイプ	テストステータス	注記
ACME	完全にテスト済み	標準の ACME 実装で検証しました。
CA	完全にテスト済み	基本的な CA 機能を確認しました。
自己署名	完全にテスト済み	基本的な自己署名機能を確認しました。
Vault	完全にテスト済み	インフラストラクチャーのアクセス制約により、標準の Vault セットアップに制限されます。
Venafi	部分的にテスト済み	プロバイダー固有の制限が適用されます。
NCM	部分的にテスト済み	プロバイダー固有の制限が適用されます。
Google CAS	部分的にテスト済み	一般的な CA 設定と互換性があります。

注記

OpenShift Container Platform は、サードパーティーの cert-manager Operator for Red Hat OpenShift プロバイダー機能に関連する要素をすべてテストするわけではありません。サードパーティーのサポートの詳細は、OpenShift Container Platform サードパーティーサポートポリシー を参照してください。

9.1.3. 証明書のリクエスト方法

cert-manager Operator for Red Hat OpenShift を使用して証明書をリクエストするには、2 つの方法があります。

cert-manager.io/CertificateRequest オブジェクトの使用

このメソッドを使用すると、サービス開発者は、設定された発行者 (サービスインフラストラクチャー管理者によって設定された) を指す有効な **issuerRef** を使用して **CertificateRequest** オブジェクトを作成します。次に、サービスインフラストラクチャー管理者は、証明書リクエストを受け入れるか拒否します。受け入れられた証明書リクエストのみが、対応する証明書を作成します。

cert-manager.io/Certificate オブジェクトの使用

このメソッドを使用すると、サービス開発者は有効な **issuerRef** を使用して **Certificate** オブジェクトを作成し、**Certificate** オブジェクトを指しているシークレットから証明書を取得します。

9.1.4. サポートされている cert-manager Operator for Red Hat OpenShift のバージョン

各 OpenShift Container Platform リリースの cert-manager Operator for Red Hat OpenShift のサポート対象バージョンに関するリストについては、OpenShift Container Platform の更新およびサポートポリシー の「Platform Agnostic Operators」セクションを参照してください。

9.1.5. cert-manager Operator for Red Hat OpenShift の FIPS 準拠について

cert-manager Operator for Red Hat OpenShift は、バージョン 1.14.0 以降で FIPS に準拠するよう設計されています。OpenShift Container Platform 上で FIPS モードで実行する場合、x86_64、ppc64le、および s390X アーキテクチャー上での FIPS 検証のために、NIST に送信された暗号化ライブラリーが使用されます。NIST 検証プログラムの詳細は、Cryptographic module validation program を参照してください。RHEL 暗号化ライブラリーの個別バージョンに関して検証用に提出された最新の NIST ステータスについては、Compliance activities and government standards を参照してください。

FIPS モードを有効にするには、FIPS モードで稼働するように設定された OpenShift Container Platform クラスター上に cert-manager Operator for Red Hat OpenShift をインストールする必要があります。詳細は、「クラスターに追加のセキュリティーが必要な場合」を参照してください。

9.1.6. 関連情報

- cert-manager project documentation
- コンプライアンスについて
- FIPS モードでのクラスターのインストール
- クラスターに追加のセキュリティーが必要ですか？

9.2. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT リリースノート

cert-manager Operator for Red Hat OpenShift は、アプリケーション証明書のライフサイクル管理を提供するクラスター全体のサービスです。

以下のリリースノートは、cert-manager Operator for Red Hat OpenShift の開発履歴を記録したものです。

詳細は、cert-manager Operator for Red Hat OpenShift について を参照してください。

9.2.1. cert-manager Operator for Red Hat OpenShift 1.17.0

発行日: 2025 年 8 月 6 日

Operator for Red Hat OpenShift 1.17.0 では、次のアドバイザリーが利用可能です。

- RHBA-2025:13182
- RHBA-2025:13134
- RHBA-2025:13133

cert-manager Operator for Red Hat OpenShift のバージョン **1.17.0** は、アップストリームの cert-manager バージョン **v1.17.4** をベースにしています。詳細は、cert-manager project release notes for v1.17.4 を参照してください。

9.2.1.1. バグ修正

- 以前は、Istio-CSR のデプロイメントが成功した後でも、**IstioCSR** カスタムリソース (CR) の **status** フィールドが **Ready** に設定されませんでした。この修正により、**status** フィールドが **Ready** に正しく設定され、ステータス報告の一貫性と信頼性が確保されます。(CM-546)

9.2.1.2. 新機能および機能拡張

ACME HTTP-01 ソルバー Pod のリソース要求と制限の設定をサポートする

このリリースでは、cert-manager Operator for Red Hat OpenShift は ACME HTTP-01 ソルバー Pod の CPU およびメモリーリソース要求と制限の設定をサポートします。**CertManager** カスタムリソース (CR) で次のオーバーライド可能な引数を使用して、CPU およびメモリーリソースの要求と制限を設定できます。

- **--acme-http01-solver-resource-limits-cpu**
- **--acme-http01-solver-resource-limits-memory**
- **--acme-http01-solver-resource-request-cpu**
- **--acme-http01-solver-resource-request-memory**

詳細は、cert-manager コンポーネントのオーバーライド可能な引数 を参照してください。

9.2.1.3. CVE

- CVE-2025-22866
- CVE-2025-22868
- CVE-2025-22872
- CVE-2025-22870
- CVE-2025-27144
- CVE-2025-22871

9.2.2. cert-manager Operator for Red Hat OpenShift 1.16.2

発行日: 2025 年 10 月 16 日

cert-manager Operator for Red Hat OpenShift 1.16.2 では、次のアドバイザリーが利用可能です。

- RHSA-2025:18219
- RHBA-2025:18223
- RHBA-2025:18229

cert-manager Operator for Red Hat OpenShift のバージョン **1.16.2** は、アップストリームの cert-manager バージョン **v1.16.5** ベースにしています。詳細は、cert-manager project release notes for v1.16.5 を参照してください。

9.2.2.1. CVE

- CVE-2024-12718
- CVE-2025-4138
- CVE-2025-4517
- CVE-2025-49794
- CVE-2025-49796
- CVE-2025-5914
- CVE-2025-6020
- CVE-2025-6965
- CVE-2025-7425
- CVE-2025-8941

9.2.3. cert-manager Operator for Red Hat OpenShift 1.16.1

発行日: 2025 年 7 月 10 日

Operator for Red Hat OpenShift 1.16.1 では、次のアドバイザリーが利用可能です。

- RHBA-2025:10765
- RHBA-2025:10766
- RHBA-2025:10785

cert-manager Operator for Red Hat OpenShift のバージョン **1.16.1** は、アップストリームの cert-manager バージョン **v1.16.5** をベースにしています。詳細は、cert-manager project release notes for v1.16.5 を参照してください。

9.2.3.1. バグ修正

以前は、cert-manager Operator for Red Hat OpenShift は、RBAC 権限が不十分なため、**cert-manager-tokenrequest** ロールを作成できませんでした。その結果、**RoleCreateFailed** エラーが発生し、静的リソースコントローラーがデグレード状態になっていました。このリリースでは、必要な **serviceaccounts/token** の作成権限を RBAC 設定に追加することで、この問題が解決されています。その結果、**cert-manager-tokenrequest** ロールおよびロールバインディングが正常に作成され、Operator ログに **RoleCreateFailed** エラーが表示されなくなりました。(OCPBUGS-56758)

9.2.3.2. CVE

- CVE-2025-22871
- CVE-2025-30204
- CVE-2025-22872
- CVE-2025-27144
- CVE-2025-22870

9.2.4. cert-manager Operator for Red Hat OpenShift 1.16.0

発行日: 2025 年 5 月 27 日

cert-manager Operator for Red Hat OpenShift 1.16.0 では、次のアドバイザリーが利用可能です。

- RHEA-2025:8163
- RHEA-2025:8164
- RHEA-2025:8165
- RHEA-2025:8198

cert-manager Operator for Red Hat OpenShift のバージョン **1.16.0** は、アップストリームの cert-manager バージョン **v1.16.4** をベースにしています。詳細は、cert-manager project release notes for v1.16.4 を参照してください。

9.2.4.1. 新機能および機能拡張

非接続環境のサポート

このリリースでは、cert-manager Operator for Red Hat OpenShift を非接続環境にミラーリングおよびインストールできることが検証済みです。

また、この Operator が非接続環境で各種の発行者 (ACME、CA、Self-signed、および Vault) と連携して動作することも検証済みです。具体的には、プライベートまたは自己ホスト型の ACME サーバーが検証済みです。非接続環境では、Let's Encrypt やその他のパブリック ACME サービスは実行可能な選択肢ではないためです。Operator イメージのミラーリング方法としては、oc-mirror プラグイン v2 が推奨されます。詳細は、oc-mirror プラグイン v2 を使用した非接続インストールのイメージのミラーリング を参照してください。

オペランドメトリクスのサポート拡張

このリリースでは、cert-manager の webhook および cainjector オペランドが、**/metrics** サービスエンドポイントを介して、デフォルトでポート 9402 で Prometheus メトリクスを公開するようになりました。組み込みのユーザーワークロードモニタリングスタックを有効にすることで、すべての cert-manager オペランドからメトリクスを収集するように OpenShift Monitoring を設定できます。詳細は、cert-manager Operator for Red Hat OpenShift のモニタリング を参照してください。

Streaming Lists の有効化

このリリースでは、cert-manager Operator for Red Hat OpenShift が、アップストリームの新機能である WatchListClient を使用するようになりました。これにより、Kubernetes API サーバーの Streaming Lists 機能が使用できるようになり、API サーバーの負荷が軽減されます。OpenShift Container Platform 4.14 以降では、cert-manager コンポーネント起動時のピーク時のメモリー使用量が最適化されています。

9.2.4.2. CVE

- CVE-2024-45337
- CVE-2024-45338
- CVE-2025-22866

9.2.4.3. 既知の問題

cert-manager バージョン 1.16.0 でユーザー名とパスワードの認証を使用して Venafi 発行者を使用する場合、デフォルトのクライアント ID が **cert-manager.io** としてハードコードされており、カスタマイズできません。この制限は、Venafi プラットフォームでの認証に特定のクライアント ID を必要とするユーザーに影響する可能性があります。

9.2.5. cert-manager Operator for Red Hat OpenShift 1.15.1

発行日: 2025 年 3 月 13 日

cert-manager Operator for Red Hat OpenShift 1.15.1 では、次のアドバイザリーが利用可能です。

- RHEA-2025-2733
- RHEA-2025-2780
- RHEA-2025-2821
- RHEA-2025-2828

cert-manager Operator for Red Hat OpenShift のバージョン **1.15.1** は、アップストリームの cert-manager バージョン **v1.15.5** をベースにしています。詳細は、cert-manager project release notes for v1.15.5 を参照してください。

9.2.5.1. 新機能および機能拡張

cert-manager Operator for Red Hat OpenShift と Istio-CSR の統合(テクノロジープレビュー)

cert-manager Operator for Red Hat OpenShift が Istio-CSR をサポートするようになりました。この統合により、cert-manager Operator の発行者が、相互 TLS (mTLS) 通信用の証明書を発行、署名、更新できるようになります。この証明書を、Red Hat OpenShift Service Mesh および **Istio** が cert-manager Operator から直接要求できるようになりました。

詳細は、cert-manager Operator と Istio-CSR の統合 を参照してください。

9.2.5.2. CVE

- CVE-2024-9287
- CVE-2024-45336
- CVE-2024-45341

9.2.6. cert-manager Operator for Red Hat OpenShift 1.15.0

発行日: 2025 年 1 月 22 日

cert-manager Operator for Red Hat OpenShift 1.15.0 では、次のアドバイザリーが利用可能です。

- RHEA-2025:0487
- RHSA-2025:0535
- RHSA-2025:0536

cert-manager Operator for Red Hat OpenShift のバージョン **1.15.0** は、アップストリームの cert-manager バージョン **v1.15.4** をベースにしています。詳細は、cert-manager project release notes for v1.15.4 を参照してください。

9.2.6.1. 新機能および機能拡張

cert-manager Operator for Red Hat OpenShift のスケジュールオーバーライド

このリリースでは、cert-manager コントローラー、webhook、CA インジェクターなどを含む cert-manager Operator for Red Hat OpenShift のスケジュールオーバーライドを設定できます。

Google CAS 発行者

cert-manager Operator for Red Hat OpenShift は、Google Certificate Authority Service (CAS) 発行者をサポートするようになりました。**google-cas-issuer** は、CAS により管理されるプライベート認証局を使用して、発行と更新を含む証明書ライフサイクルの管理を自動化する cert-manager の外部発行者です。

> **注記**
>
> Google CAS 発行者は、バージョン 0.9.0 および cert-manager Operator for Red Hat OpenShift バージョン 1.15.0 でのみ検証されます。これらのバージョンは、OpenShift Container Platform クラスターの API サーバーおよび Ingress コントローラーの証明書の発行、更新、管理などのタスクをサポートします。

デフォルトの installMode を AllNamespaces に更新

バージョン 1.15.0 以降では、Operator Lifecycle Manager (OLM) のデフォルトかつ推奨される **installMode** は **AllNamespaces** です。以前のデフォルトは **SingleNamespace** でした。この変更は、multi-namespace Operator 管理のベストプラクティスと一致しています。詳細は、OCPBUGS-23406 を参照してください。

冗長 kube-rbac-proxy サイドカーの削除

Operator に、冗長な **kube-rbac-proxy** サイドカーコンテナーがなくなり、リソースの使用量と複雑さが軽減されました。詳細は、CM-436 を参照してください。

9.2.6.2. CVE

- CVE-2024-35255
- CVE-2024-28180
- CVE-2024-24783
- CVE-2024-6104
- CVE-2023-45288
- CVE-2024-45337
- CVE-2024-45338

9.2.7. cert-manager Operator for Red Hat OpenShift 1.16.2

発行日: 2025 年 10 月 16 日

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.14.1:

- RHEA-2025:8163
- RHEA-2025:8163
- RHSA-2025:18219

cert-manager Operator for Red Hat OpenShift のバージョン **1.16.2** は、アップストリームの cert-manager バージョン **v1.16.5** ベースにしています。詳細は、cert-manager project release notes for v1.14.7 を参照してください。

9.2.7.1. 新機能および機能拡張

冗長 kube-rbac-proxy サイドカーの削除

Operator に、冗長な **kube-rbac-proxy** サイドカーコンテナーがなくなり、リソースの使用量と複雑さが軽減されました。詳細は、CM-436 を参照してください。

9.2.7.2. CVE

- CVE-2024-6104
- CVE-2024-41110
- CVE-2025-22870
- CVE-2024-45338

9.2.8. cert-manager Operator for Red Hat OpenShift 1.14.1

発行日: 2024 年 11 月 4 日

The following advisory is available for the cert-manager Operator for Red Hat OpenShift 1.14.1:

- RHEA-2024:8787

cert-manager Operator for Red Hat OpenShift のバージョン **1.14.1** は、アップストリームの cert-manager バージョン **v1.14.7** をベースにしています。詳細は、cert-manager project release notes for v1.14.7 を参照してください。

9.2.8.1. CVE

- CVE-2024-33599
- CVE-2024-2961

9.2.9. cert-manager Operator for Red Hat OpenShift 1.14.0

発行日: 2024 年 7 月 8 日

cert-manager Operator for Red Hat OpenShift 1.14.0 については、次のアドバイザリーが利用できます。

- RHEA-2024:4360

cert-manager Operator for Red Hat OpenShift のバージョン **1.14.0** は、アップストリームの cert-manager バージョン **v1.14.5** をベースにしています。詳細は、cert-manager project release notes for v1.14.5 を参照してください。

9.2.9.1. 新機能および機能拡張

FIPS 準拠のサポート

このリリースでは、cert-manager Operator for Red Hat OpenShift に対して FIPS モードが自動的に有効化されるようになりました。FIPS モードの OpenShift Container Platform クラスターにインストールすると、cert-manager Operator for Red Hat OpenShift はクラスターの FIPS サポートステータスに影響を与えることなく互換性を確保します。

cert-manager が管理する証明書によるルートの保護 (テクノロジープレビュー)

このリリースでは、cert-manager Operator for Red Hat OpenShift を使用して、**Route** リソースで参照される証明書を管理できます。詳細は、cert-manager Operator for Red Hat OpenShift によるルート保護 を参照してください。

NCM 発行者

cert-manager Operator for Red Hat OpenShift は、Nokia NetGuard Certificate Manager (NCM) 発行者をサポートします。**ncm-issuer** は、Kubernetes コントローラーを使用して NCM PKI システムと統合し、証明書要求に署名する cert-manager 外部発行者です。この統合により、アプリケーションの非自己署名付き証明書の取得プロセスが合理化され、証明書の有効性と最新の状態が確保されます。

> **注記**
>
> NCM 発行者は、バージョン 1.1.1 および cert-manager Operator for Red Hat OpenShift バージョン 1.14.0 に対してのみ検証されます。このバージョンは、OpenShift Container Platform クラスターの API サーバーおよび Ingress コントローラーの証明書の発行、更新、管理などのタスクを処理します。

9.2.9.2. CVE

- CVE-2023-45288
- CVE-2024-28180
- CVE-2020-8559
- CVE-2024-26147
- CVE-2024-24783

9.2.10. cert-manager Operator for Red Hat OpenShift 1.13.1

発行日: 2024 年 5 月 15 日

cert-manager Operator for Red Hat OpenShift 1.13.1 については、次のアドバイザリーが利用できます。

- RHEA-2024:2849

cert-manager Operator for Red Hat OpenShift のバージョン **1.13.1** は、アップストリームの cert-manager バージョン **v1.13.6** をベースにしています。詳細は、cert-manager project release notes for v1.13.6 を参照してください。

9.2.10.1. CVE

- CVE-2023-45288
- CVE-2023-48795
- CVE-2024-24783

9.2.11. cert-manager Operator for Red Hat OpenShift 1.13.0

発行日: 2024 年 1 月 16 日

cert-manager Operator for Red Hat OpenShift 1.13.0 については、次のアドバイザリーが利用できます。

- RHEA-2024:0259

cert-manager Operator for Red Hat OpenShift のバージョン **1.13.0** は、アップストリームの cert-manager バージョン **v1.13.3** をベースにしています。詳細は、cert-manager project release notes for v1.13.0 を参照してください。

9.2.11.1. 新機能および機能拡張

- cert-manager Operator for Red Hat OpenShift を使用して、API サーバーと Ingress コントローラーの証明書を管理できるようになりました。詳細は、発行者による証明書の設定 を参照してください。
- このリリースでは、以前は AMD64 アーキテクチャー上の OpenShift Container Platform に限定されていた cert-manager Operator for Red Hat OpenShift の範囲が拡張され、IBM Z® (**s390x**)、IBM Power® (**ppc64le**)、ARM64 アーキテクチャー上で稼働する OpenShift Container Platform 上での証明書管理も対象に含まれるようになりました。
- このリリースでは、ACME DNS-01 チャレンジ認証中にセルフチェックを実行するために DNS over HTTPS (DoH) を使用できるようになりました。DNS セルフチェック方式は、コマンドラインフラグ **--dns01-recursive-nameservers-only** および **--dns01-recursive-nameservers** を使用して制御できます。詳細は、cert-manager Operator API からの引数をオーバーライドして cert-manager をカスタマイズする を参照してください。

9.2.11.2. CVE

- CVE-2023-39615
- CVE-2023-3978
- CVE-2023-37788
- CVE-2023-29406

9.3. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT のインストール

重要

cert-manager Operator for Red Hat OpenShift バージョン 1.15 以降では、**AllNamespaces**、**SingleNamespace**、**OwnNamespace** インストールモードがサポートされます。それより前のバージョン (例: 1.14) では、**SingleNamespace** および **OwnNamespace** インストールモードのみサポートされます。

cert-manager Operator for Red Hat OpenShift は、デフォルトでは OpenShift Container Platform にインストールされません。Web コンソールを使用して、cert-manager Operator for Red Hat OpenShift をインストールできます。

9.3.1. cert-manager Operator for Red Hat OpenShift のインストール

9.3.1.1. Web コンソールを使用した cert-manager Operator for Red Hat OpenShift のインストール

Web コンソールを使用して、cert-manager Operator for Red Hat OpenShift をインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. **Operators → OperatorHub** に移動します。
3. フィルターボックスに **cert-manager Operator for Red Hat OpenShift**と入力します。
4. **cert-manager Operator for Red Hat OpenShift**を選択します。
5. **Version** ドロップダウンリストから cert-manager Operator for Red Hat OpenShift バージョンを選択し、**Install** をクリックします。

   注記

   サポートされる cert-manager Operator for Red Hat OpenShift バージョンについては、以下の「関連情報」セクションを参照してください。

6. **Install Operator** ページで以下を行います。
   a. 必要に応じて、**Update channel** を更新します。チャネルはデフォルトで **stable-v1** に設定されます。これに設定すると、cert-manager Operator for Red Hat OpenShift の最新の安定版リリースがインストールされます。
   b. Operator の **Installed Namespace** を選択します。デフォルトの Operator namespace は **cert-manager-operator** です。
   cert-manager-operator namespace は、存在しない場合、作成されます。

注記

インストール中に、OpenShift Container Platform Web コンソールを使用して、**AllNamespaces** および **SingleNamespace** インストールモードのいずれかを選択できます。cert-manager Operator for Red Hat OpenShift version 1.15.0 以降を使用したインストールでは、**AllNamespaces** インストールモードを選択することが推奨されます。それより前のバージョンでは **SingleNamespace** および **OwnNamespace** のサポートが維持されますが、今後のバージョンでは非推奨になります。

c. **Update approval** ストラテジーを選択します。

- **Automatic** ストラテジーを使用すると、新しいバージョンが利用可能になったときに、Operator Lifecycle Manager (OLM) によって Operator を自動的に更新できます。
- **Manual** ストラテジーには、Operator の更新を承認するための適切な認証情報を持つユーザーが必要です。

d. **Install** をクリックします。

検証

1. **Operators → Installed Operators** に移動します。
2. **cert-manager Operator for Red Hat OpenShift**が、**cert-manager-operator** namespace に **ステータス Succeeded** でリストされていることを確認します。
3. 次のコマンドを入力して、cert-manager Pod が動作していることを確認します。

```
$ oc get pods -n cert-manager
```

出力例

```
NAME                                   READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt             1/1     Running   0          3m39s
cert-manager-cainjector-56cc5f9868-7g9z7   1/1     Running   0          4m5s
cert-manager-webhook-d4f79d7f7-9dg9w      1/1     Running   0          4m9s
```

cert-manager Operator for Red Hat OpenShift は、cert-manager Pod が起動して実行された後でなければ使用できません。

9.3.1.2. CLI を使用した cert-manager Operator for Red Hat OpenShift のインストール

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。

手順

1. 次のコマンドを実行して、**cert-manager-operator** という名前の新規プロジェクトを作成します。

```
$ oc new-project cert-manager-operator
```

2. **OperatorGroup** オブジェクトを作成します。

   a. 以下の内容で YAML ファイル (例: **operatorGroup.yaml**) を作成します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  targetNamespaces:
  - "cert-manager-operator"
```

   b. cert-manager Operator for Red Hat OpenShift v1.15.0 以降の場合、次の内容の YAML ファイルを作成します。

```
apiVersion: operators.coreos.com/v1
kind: OperatorGroup
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  targetNamespaces: []
  spec: {}
```

注記

cert-manager Operator for Red Hat OpenShift バージョン 1.15.0 以降では、**AllNamespaces** OLM **installMode** を使用して Operator をインストールすることが推奨されます。古いバージョンでは、引き続き **SingleNamespace** または **OwnNamespace** の OLM **installMode** を使用できます。**SingleNamespace** および **OwnNamespace** のサポートは、今後のバージョンで非推奨になります。

   c. 以下のコマンドを実行して **OperatorGroup** オブジェクトを作成します。

```
$ oc create -f operatorGroup.yaml
```

3. **Subscription** オブジェクトを作成します。

   a. **Subscription** オブジェクトを定義する YAML ファイル (例: **subscription.yaml**) を作成します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: openshift-cert-manager-operator
  namespace: cert-manager-operator
spec:
  channel: stable-v1
  name: openshift-cert-manager-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  installPlanApproval: Automatic
```

-

b. 以下のコマンドを実行して **Subscription** オブジェクトを作成します。

```
$ oc create -f subscription.yaml
```

検証

1. 次のコマンドを実行して、OLM サブスクリプションが作成されていることを確認します。

```
$ oc get subscription -n cert-manager-operator
```

出力例

```
NAME                            PACKAGE                         SOURCE           CHANNEL
openshift-cert-manager-operator   openshift-cert-manager-operator   redhat-operators
stable-v1
```

2. 次のコマンドを実行して、Operator が正常にインストールされているか確認します。

```
$ oc get csv -n cert-manager-operator
```

出力例

```
NAME                          DISPLAY                                   VERSION   REPLACES
PHASE
cert-manager-operator.v1.13.0   cert-manager Operator for Red Hat OpenShift   1.13.0
cert-manager-operator.v1.12.1   Succeeded
```

3. 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のステータスが **Running** であることを確認します。

```
$ oc get pods -n cert-manager-operator
```

出力例

```
NAME                                                    READY   STATUS    RESTARTS   AGE
cert-manager-operator-controller-manager-695b4d46cb-r4hld   2/2     Running   0          7m4s
```

4. 次のコマンドを実行して、cert-manager Pod のステータスが **Running** であることを確認します。

```
$ oc get pods -n cert-manager
```

出力例

```
NAME                                       READY   STATUS    RESTARTS   AGE
cert-manager-58b7f649c4-dp6l4              1/1     Running   0          7m1s
cert-manager-cainjector-5565b8f897-gx25h   1/1     Running   0          7m37s
cert-manager-webhook-9bc98cbdd-f972x       1/1     Running   0          7m40s
```

関連情報

- サポートされている cert-manager Operator for Red Hat OpenShift のバージョン

9.3.2. cert-manager Operator for Red Hat OpenShift の更新チャネルについて

更新チャネルは、クラスター内の cert-manager Operator for Red Hat OpenShift のバージョンを宣言できるメカニズムです。cert-manager Operator for Red Hat OpenShift では、次の更新チャネルを使用できます。

- **stable-v1**
- **stable-v1.y**

9.3.2.1. stable-v1 チャネル

stable-v1 チャネルは、cert-manager Operator for Red Hat OpenShift の最新リリースバージョンをインストールおよび更新します。cert-manager Operator for Red Hat OpenShift の最新の stable リリースを使用する場合は、**stable-v1** チャネルを選択します。

注記

stable-v1 チャネルは、cert-manager Operator for Red Hat OpenShift をインストールする際のデフォルトの推奨チャネルです。

stable-v1 チャネルでは、次の更新承認ストラテジーを使用できます。

自動

インストールされている cert-manager Operator for Red Hat OpenShift の自動更新を選択した場合、新しいバージョンの cert-manager Operator for Red Hat OpenShift が **stable-v1** チャネルで利用可能になります。Operator Lifecycle Manager (OLM) は、人間の介入なしに Operator の実行中のインスタンスを自動的にアップグレードします。

Manual

手動更新を選択した場合、cert-manager Operator for Red Hat OpenShift の新しいバージョンが利用可能になると、OLM は更新リクエストを作成します。クラスター管理者は、その更新リクエストを手動で承認して、cert-manager Operator for Red Hat OpenShift を新しいバージョンに更新する必要があります。

9.3.2.2. stable-v1.y チャネル

cert-manager Operator for Red Hat OpenShift の y-stream バージョンは、**stable-v1.10**、**stable-v1.11**、**stable-v1.12** などの **stable-v1.y** チャネルから更新をインストールします。y-stream バージョンを使用し、cert-manager Operator for Red Hat OpenShift の最新の z-stream バージョンを保つ場合は、**stable-v1.y** チャネルを選択します。

stable-v1.y チャネルでは、次の更新承認ストラテジーを使用できます。

自動

インストールされている cert-manager Operator for Red Hat OpenShift の自動更新を選択した場合、cert-manager Operator for Red Hat OpenShift の新しい z-stream バージョンが **stable-v1.y** チャネルで利用可能になります。OLM は、人間の介入なしで Operator の実行中のインスタンスを自動的にアップグレードします。

Manual

手動更新を選択した場合、cert-manager Operator for Red Hat OpenShift の新しいバージョンが利用可能になると、OLM は更新リクエストを作成します。クラスター管理者は、その更新リクエストを手動で承認して、cert-manager Operator for Red Hat OpenShift を新しい z-stream リリースバージョンに更新する必要があります。

9.3.3. 関連情報

- クラスターへの Operator の追加
- インストール済み Operator の更新

9.4. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT の EGRESS プロキシーの設定

OpenShift Container Platform でクラスター全体の Egress プロキシーが設定されている場合は、Operator Lifecycle Manager (OLM) が、自身が管理する Operator に、そのクラスター全体のプロキシーを自動的に設定します。OLM は、Operator のすべてのデプロイメントを **HTTP_PROXY**、**HTTPS_PROXY**、**NO_PROXY** 環境変数で自動的に更新します。

HTTPS 接続をプロキシーするために必要な CA 証明書を cert-manager Operator for Red Hat OpenShift に挿入できます。

9.4.1. cert-manager Operator for Red Hat OpenShift のカスタム CA 証明書の挿入

OpenShift Container Platform クラスターでクラスター全体のプロキシーが有効になっている場合は、HTTPS 接続をプロキシーするために必要な CA 証明書を cert-manager Operator for Red Hat OpenShift に挿入できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- OpenShift Container Platform のクラスター全体のプロキシーを有効にしている。

手順

1. 次のコマンドを実行して、**cert-manager** namespace に config map を作成します。

```
$ oc create configmap trusted-ca -n cert-manager
```

2. 以下のコマンドを実行して、OpenShift Container Platform によって信頼されている CA バンドルを config map に挿入します。

```
$ oc label cm trusted-ca config.openshift.io/inject-trusted-cabundle=true -n cert-manager
```

3. 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のデプロイメントを更新し、config map を使用します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --
type='merge' -p '{"spec":{"config":{"env":
[{"name":"TRUSTED_CA_CONFIGMAP_NAME","value":"trusted-ca"}]}}}'
```

検証

1. 次のコマンドを実行して、デプロイメントのロールアウトが完了したことを確認します。

```
$ oc rollout status deployment/cert-manager-operator-controller-manager -n cert-manager-operator && \
oc rollout status deployment/cert-manager -n cert-manager && \
oc rollout status deployment/cert-manager-webhook -n cert-manager && \
oc rollout status deployment/cert-manager-cainjector -n cert-manager
```

出力例

```
deployment "cert-manager-operator-controller-manager" successfully rolled out
deployment "cert-manager" successfully rolled out
deployment "cert-manager-webhook" successfully rolled out
deployment "cert-manager-cainjector" successfully rolled out
```

2. 次のコマンドを実行して、CA バンドルがボリュームとしてマウントされたことを確認します。

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.'containers[0].volumeMounts'}
```

出力例

```
[{"mountPath":"/etc/pki/tls/certs/cert-manager-tls-ca-bundle.crt","name":"trusted-ca","subPath":"ca-bundle.crt"}]
```

3. 次のコマンドを実行して、CA バンドルのソースが **trusted-ca** config map であることを確認します。

```
$ oc get deployment cert-manager -n cert-manager -o=jsonpath=
{.spec.template.spec.volumes}
```

出力例

```
[{"configMap":{"defaultMode":420,"name":"trusted-ca"},"name":"trusted-ca"}]
```

9.4.2. 関連情報

- Operator Lifecycle Manager でのプロキシーサポートの設定

9.5. CERTMANAGER カスタムリソースを使用して CERT-MANAGER OPERATOR をカスタマイズする

cert-manager Operator for Red Hat OpenShift をインストールした後、**CertManager** カスタムリソース (CR) を設定することで、次のアクションを実行できます。

- cert-manager コントローラー、CA インジェクター、Webhook などの cert-manager コンポーネントの動作を変更するには、引数を設定します。
- コントローラー Pod の環境変数を設定します。

- CPU とメモリーの使用量を管理するために、リソース要求と制限を定義します。
- クラスター内で Pod が実行される場所を制御するためのスケジューリングルールを設定します。

CertManager CR YAML ファイルの例

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
spec:
  controllerConfig:
    overrideArgs:
      - "--dns01-recursive-nameservers=8.8.8.8:53,1.1.1.1:53"
    overrideEnv:
      - name: HTTP_PROXY
        value: http://proxy.example.com:8080
    overrideResources:
      limits:
        cpu: "200m"
        memory: "512Mi"
      requests:
        cpu: "100m"
        memory: "256Mi"
    overrideScheduling:
      nodeSelector:
        custom: "label"
      tolerations:
        - key: "key1"
          operator: "Equal"
          value: "value1"
          effect: "NoSchedule"

  webhookConfig:
    overrideArgs:
#...
    overrideResources:
#...
    overrideScheduling:
#...

  cainjectorConfig:
    overrideArgs:
#...
    overrideResources:
#...
    overrideScheduling:
#...
```

警告

サポートされていない引数をオーバーライドするには、**CertManager** リソースに **spec.unsupportedConfigOverrides** セクションを追加しますが、**spec.unsupportedConfigOverrides** の使用はサポートされていません。

9.5.1. CertManager カスタムリソースのフィールドの説明

CertManager カスタムリソース (CR) を使用して、cert-manager Operator for Red Hat OpenShift の次のコアコンポーネントを設定できます。

- cert-manager コントローラー: **spec.controllerConfig** フィールドを使用して、cert-manager コントローラー Pod を設定できます。
- Webhook: **spec.webhookConfig** フィールドを使用して、検証および変更リクエストを処理する Webhook Pod を設定できます。
- CA インジェクター: **spec.cainjectorConfig** フィールドを使用して、CA インジェクター Pod を設定できます。

9.5.1.1. cert-manager コンポーネントの CertManager CR における共通の設定可能フィールド

次の表は、**CertManager** CR の **spec.controllerConfig**、**spec.webhookConfig**、および **spec.cainjectorConfig** セクションで設定できる共通フィールドを示しています。

表9.1 cert-manager コンポーネントの CertManager CR における共通の設定可能フィールド

フィールド	型	説明
overrideArgs	**string**	cert-manager コンポーネントでサポートされている引数をオーバーライドできます。
overrideEnv	**dict**	cert-manager コントローラーでサポートされている環境変数をオーバーライドできます。このフィールドは、cert-manager コントローラーコンポーネントでのみサポートされます。
overrideResources	**object**	cert-manager コンポーネントの CPU およびメモリーの制限を設定できます。
overrideScheduling	**object**	cert-manager コンポーネントの Pod スケジューリング制約を設定できます。

9.5.1.2. cert-manager コンポーネントのオーバーライド可能な引数

CertManager CR の **spec.controllerConfig**、**spec.webhookConfig**、および **spec.cainjectorConfig** セクションで、cert-manager コンポーネントのオーバーライド可能な引数を設定できます。

次の表は、cert-manager コンポーネントのオーバーライド可能な引数について説明しています。

表9.2 cert-manager コンポーネントのオーバーライド可能な引数

引数	Component	説明
--dns01-recursive-nameservers=<server_address>	Controller	DNS-01 セルフチェックをクエリーするネームサーバーのコンマ区切りリストを指定します。ネームサーバーは、**<host>:<port>** (例: **1.1.1.1:53**) で指定することも、DNS over HTTPS (DoH) (例: **https://1.1.1.1/dns-query**) を使用することもできます。 **注記** DNS over HTTPS (DoH) は、cert-manager Operator for Red Hat OpenShift バージョン 1.13.0 以降でのみサポートされます。
--dns01-recursive-nameservers-only	Controller	そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。
--acme-http01-solver-nameservers=<host>:<port>	Controller	Automated Certificate Management Environment (ACME) HTTP01 セルフチェックをクエリーするための **<host>:<port>** ネームサーバーをコンマ区切りのリストで指定します。たとえば **--acme-http01-solver-nameservers=1.1.1.1:53** です。
--metrics-listen-address=<host>:<port>	Controller	メトリクスエンドポイントのホストとポートを指定します。デフォルト値は **--metrics-listen-address=0.0.0.0:9402** です。
--issuer-ambient-credentials	Controller	この引数を使用すると、アンビエント認証情報を使用して DNS-01 チャレンジを解決するように ACME Issuer を設定できます。

引数	Component	説明
--enable-certificate-owner-ref	Controller	この引数は、証明書リソースを、TLS 証明書が保存されているシークレットの所有者として設定します。詳細は、「証明書の削除時に TLS シークレットを自動的に削除する」を参照してください。
--acme-http01-solver-resource-limits-cpu	Controller	ACME HTTP-01 ソルバー Pod の最大 CPU 制限を定義します。デフォルト値は **100m** です。
--acme-http01-solver-resource-limits-memory	Controller	ACME HTTP-01 ソルバー Pod の最大メモリー制限を定義します。デフォルト値は **64Mi** です。
--acme-http01-solver-resource-request-cpu	Controller	ACME HTTP-01 ソルバー Pod の最小 CPU 要求を定義します。デフォルト値は **10m** です。
--acme-http01-solver-resource-request-memory	Controller	ACME HTTP-01 ソルバー Pod の最小メモリー要求を定義します。デフォルト値は **64Mi** です。
--v=<verbosity_level>	コントローラー、Webhook、CA インジェクター	ログメッセージの冗長性を決定するために、ログレベルの詳細度を指定します。

9.5.1.3. cert-manager コントローラーのオーバーライド可能な環境変数

CertManager CR の **spec.controllerConfig.overrideEnv** フィールドで、cert-manager コントローラーのオーバーライド可能な環境変数を設定できます。

次の表は、cert-manager コントローラーのオーバーライド可能な環境変数について説明しています。

表9.3 cert-manager コントローラーのオーバーライド可能な環境変数

Environment variable	説明
HTTP_PROXY	送信 HTTP 要求のプロキシーサーバー。
HTTPS_PROXY	送信 HTTPS 要求のプロキシーサーバー。
NO_PROXY	プロキシーをバイパスするホストのコンマ区切りリスト。

9.5.1.4. cert-manager コンポーネントのオーバーライド可能なリソースパラメーター

CertManager CR の **spec.controllerConfig**、**spec.webhookConfig**、および **spec.cainjectorConfig** セクションで、cert-manager コンポーネントの CPU およびメモリー制限を設定できます。

次の表は、cert-manager コンポーネントのオーバーライド可能なリソースパラメーターについて説明しています。

表9.4 cert-manager コンポーネントのオーバーライド可能なリソースパラメーター

フィールド	説明
overrideResources.limits.cpu	コンポーネント Pod が使用できる CPU の最大量を定義します。
overrideResources.limits.memory	コンポーネント Pod が使用できるメモリーの最大量を定義します。
overrideResources.requests.cpu	コンポーネント Pod のスケジューラーによって要求される CPU の最小量を定義します。
overrideResources.requests.memory	コンポーネント Pod のスケジューラーによって要求されるメモリーの最小量を定義します。

9.5.1.5. cert-manager コンポーネントのオーバーライド可能なスケジューリングパラメーター

CertManager CR の **spec.controllerConfig**、**spec.webhookConfig** フィールド、および **spec.cainjectorConfig** セクションで、cert-manager コンポーネントの Pod スケジューリング制約を設定できます。

次の表は、cert-manager コンポーネントの Pod スケジューリングパラメーターについて説明しています。

表9.5 cert-manager コンポーネントのオーバーライド可能なスケジューリングパラメーター

フィールド	説明
overrideScheduling.nodeSelector	Pod を特定のノードに制限するためのキーと値のペア。
overrideScheduling.tolerations	taint されたノードで Pod をスケジュールするための toleration リスト。

関連情報

- 証明書の削除時に TLS シークレットを自動的に削除する

9.5.2. cert-manager Operator API から環境変数をオーバーライドして cert-manager をカスタマイズ

CertManager リソースに **spec.controllerConfig** セクションを追加することで、cert-manager Operator for Red Hat OpenShift でサポートされている環境変数をオーバーライドできます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできる。

手順

1. 次のコマンドを実行して、**CertManager** リソースを編集します。

```
$ oc edit certmanager cluster
```

2. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
  ...
spec:
  ...
  controllerConfig:
    overrideEnv:
      - name: HTTP_PROXY
        value: http://<proxy_url> 1
      - name: HTTPS_PROXY
        value: https://<proxy_url> 2
      - name: NO_PROXY
        value: <ignore_proxy_domains> 3
```

1 2 **<proxy_url>** をプロキシーサーバーの URL に置き換えます。

3 **<ignore_proxy_domains>** をドメインのコンマ区切りリストに置き換えます。これらのドメインは、プロキシーサーバーにより無視されます。

注記

オーバーライド可能な環境変数の詳細は、「CertManager カスタムリソースのフィールドの説明」の「cert-manager コンポーネントのオーバーライド可能な環境変数」を参照してください。

3. 変更を保存してテキストエディターを終了し、変更を適用します。

検証

1. 次のコマンドを実行して、cert-manager コントローラー Pod が再デプロイされているか確認します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

出力例

```
NAME                         READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. 次のコマンドを実行して、cert-manager Pod の環境変数が更新されているか確認します。

```
$ oc get pod <redeployed_cert-manager_controller_pod> -n cert-manager -o yaml
```

出力例

```
    env:
    ...
    - name: HTTP_PROXY
      value: http://<PROXY_URL>
    - name: HTTPS_PROXY
      value: https://<PROXY_URL>
    - name: NO_PROXY
      value: <IGNORE_PROXY_DOMAINS>
```

関連情報

- CertManager カスタムリソースのフィールドの説明

9.5.3. cert-manager Operator API からの引数をオーバーライドして cert-manager をカスタマイズ

CertManager リソースに **spec.controllerConfig** セクションを追加することで、cert-manager Operator for Red Hat OpenShift でサポートされる引数をオーバーライドできます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできる。

手順

1. 次のコマンドを実行して、**CertManager** リソースを編集します。

```
$ oc edit certmanager cluster
```

2. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
  ...
spec:
  ...
  controllerConfig:
    overrideArgs:
```

```
      - '--dns01-recursive-nameservers=<server_address>' (1)
      - '--dns01-recursive-nameservers-only'
      - '--acme-http01-solver-nameservers=<host>:<port>'
      - '--v=<verbosity_level>'
      - '--metrics-listen-address=<host>:<port>'
      - '--issuer-ambient-credentials'
      - '--acme-http01-solver-resource-limits-cpu=<quantity>'
      - '--acme-http01-solver-resource-limits-memory=<quantity>'
      - '--acme-http01-solver-resource-request-cpu=<quantity>'
      - '--acme-http01-solver-resource-request-memory=<quantity>'
  webhookConfig:
    overrideArgs:
      - '--v=<verbosity_level>'
  cainjectorConfig:
    overrideArgs:
      - '--v=<verbosity_level>'
```

1. オーバーライド可能な引数の詳細は、「CertManager カスタムリソースのフィールドの説明」の「cert-manager コンポーネントのオーバーライド可能な引数」を参照してください。

3. 変更を保存してテキストエディターを終了し、変更を適用します。

検証

- 次のコマンドを実行して、cert-manager Pod の引数が更新されているか確認します。

```
$ oc get pods -n cert-manager -o yaml
```

出力例

```
...
  metadata:
    name: cert-manager-6d4b5d4c97-kldwl
    namespace: cert-manager
...
  spec:
    containers:
    - args:
      - --acme-http01-solver-nameservers=1.1.1.1:53
      - --cluster-resource-namespace=$(POD_NAMESPACE)
      - --dns01-recursive-nameservers=1.1.1.1:53
      - --dns01-recursive-nameservers-only
      - --leader-election-namespace=kube-system
      - --max-concurrent-challenges=60
      - --metrics-listen-address=0.0.0.0:9042
      - --v=6
...
  metadata:
    name: cert-manager-cainjector-866c4fd758-ltxxj
    namespace: cert-manager
...
  spec:
    containers:
```

```
    - args:
      - --leader-election-namespace=kube-system
      - --v=2
...
  metadata:
    name: cert-manager-webhook-6d48f88495-c88gd
    namespace: cert-manager
...
  spec:
    containers:
    - args:
      ...
      - --v=4
```

関連情報

- CertManager カスタムリソースのフィールドの説明

9.5.4. 証明書の削除時に TLS シークレットを自動的に削除する

CertManager リソースに **spec.controllerConfig** セクションを追加することで、cert-manager Operator for Red Hat OpenShift の **--enable-certificate-owner-ref** フラグを有効にできます。**--enable-certificate-owner-ref** フラグは、TLS 証明書が保存されているシークレットの所有者として証明書リソースを設定します。

> **警告**
>
> cert-manager Operator for Red Hat OpenShift をアンインストールするか、クラスターから証明書リソースを削除すると、シークレットは自動的に削除されます。証明書 TLS シークレットが使用されている場所によっては、これが原因でネットワーク接続の問題が発生する可能性があります。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.12.0 以降がインストールされている。

手順

1. 次のコマンドを実行して、**Certificate** オブジェクトとそのシークレットが利用可能であることを確認します。

```
$ oc get certificate
```

出力例

```
NAME                                          READY   SECRET                                    AGE
certificate-from-clusterissuer-route53-ambient   True    certificate-from-clusterissuer-route53-
ambient   8h
```

2. 次のコマンドを実行して、**CertManager** リソースを編集します。

```
$ oc edit certmanager cluster
```

3. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
# ...
spec:
# ...
  controllerConfig:
    overrideArgs:
      - '--enable-certificate-owner-ref'
```

4. 変更を保存してテキストエディターを終了し、変更を適用します。

検証

- 次のコマンドを実行して、cert-manager コントローラー Pod の **--enable-certificate-owner-ref** フラグが更新されていることを確認します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager -o yaml
```

出力例

```
# ...
  metadata:
    name: cert-manager-6e4b4d7d97-zmdnb
    namespace: cert-manager
# ...
  spec:
    containers:
    - args:
      - --enable-certificate-owner-ref
```

9.5.5. cert-manager コンポーネントの CPU およびメモリー制限をオーバーライドする

cert-manager Operator for Red Hat OpenShift をインストールした後、cert-manager コントローラー、CA インジェクター、Webhook などの cert-manager コンポーネントの cert-manager Operator for Red Hat OpenShift API から CPU およびメモリーの制限を設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできる。

- cert-manager Operator for Red Hat OpenShift のバージョン 1.12.0 以降がインストールされている。

手順

1. 次のコマンドを入力して、cert-manager コントローラー、CA インジェクター、および Webhook のデプロイメントが使用可能であることを確認します。

```
$ oc get deployment -n cert-manager
```

出力例

```
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager            1/1     1            1           53m
cert-manager-cainjector 1/1     1            1           53m
cert-manager-webhook    1/1     1            1           53m
```

2. CPU とメモリーの制限を設定する前に、次のコマンドを入力して cert-manager コントローラー、CA インジェクター、および Webhook の既存の設定を確認します。

```
$ oc get deployment -n cert-manager -o yaml
```

出力例

```
# ...
  metadata:
    name: cert-manager
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-controller
          resources: {} 1
# ...
  metadata:
    name: cert-manager-cainjector
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
        containers:
        - name: cert-manager-cainjector
          resources: {} 2
# ...
  metadata:
    name: cert-manager-webhook
    namespace: cert-manager
# ...
  spec:
    template:
      spec:
```

```
      containers:
      - name: cert-manager-webhook
        resources: {} (3)
# ...
```

1 2 3 **spec.resources** フィールドはデフォルトで空です。cert-manager コンポーネントには CPU とメモリーの制限がありません。

3. cert-manager コントローラー、CA インジェクター、Webhook の CPU およびメモリー制限を設定するには、次のコマンドを入力します。

```
$ oc patch certmanager.operator cluster --type=merge -p="
spec:
  controllerConfig:
    overrideResources: (1)
      limits:
        cpu: 200m
        memory: 64Mi
      requests:
        cpu: 10m
        memory: 16Mi
  webhookConfig:
    overrideResources:
      limits:
        cpu: 200m
        memory: 64Mi
      requests:
        cpu: 10m
        memory: 16Mi
  cainjectorConfig:
    overrideResources:
      limits:
        cpu: 200m
        memory: 64Mi
      requests:
        cpu: 10m
        memory: 16Mi
"
```

1 オーバーライド可能なリソースパラメーターの詳細は、「CertManager カスタムリソースのフィールドの説明」の「cert-manager コンポーネントのオーバーライド可能なリソースパラメーター」を参照してください。

出力例

```
certmanager.operator.openshift.io/cluster patched
```

検証

1. cert-manager コンポーネントの CPU とメモリーの制限が更新されていることを確認します。

```
$ oc get deployment -n cert-manager -o yaml
```

出力例

```
# ...
 metadata:
   name: cert-manager
   namespace: cert-manager
# ...
 spec:
   template:
     spec:
       containers:
       - name: cert-manager-controller
         resources:
           limits:
             cpu: 200m
             memory: 64Mi
           requests:
             cpu: 10m
             memory: 16Mi
# ...
 metadata:
   name: cert-manager-cainjector
   namespace: cert-manager
# ...
 spec:
   template:
     spec:
       containers:
       - name: cert-manager-cainjector
         resources:
           limits:
             cpu: 200m
             memory: 64Mi
           requests:
             cpu: 10m
             memory: 16Mi
# ...
 metadata:
   name: cert-manager-webhook
   namespace: cert-manager
# ...
 spec:
   template:
     spec:
       containers:
       - name: cert-manager-webhook
         resources:
           limits:
             cpu: 200m
             memory: 64Mi
           requests:
             cpu: 10m
             memory: 16Mi
# ...
```

関連情報

- CertManager カスタムリソースのフィールドの説明

9.5.6. cert-manager コンポーネントのスケジュールオーバーライドを設定する

cert-manager コントローラー、CA インジェクター、Webhook などの cert-manager Operator for Red Hat OpenShift コンポーネントの Pod スケジューリングを、cert-manager Operator for Red Hat OpenShift API から設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform クラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.15.0 以降がインストールされている。

手順

- 次のコマンドを実行して **certmanager.operator** カスタムリソースを更新し、目的のコンポーネントの Pod スケジューリングオーバーライドを設定します。**nodeSelector** および **tolerations** 設定を定義するには、**controllerConfig**、**webhookConfig**、または **cainjectorConfig** セクションの **overrideScheduling** フィールドを使用します。

```
$ oc patch certmanager.operator cluster --type=merge -p="
spec:
  controllerConfig:
    overrideScheduling: (1)
      nodeSelector:
        node-role.kubernetes.io/control-plane: ''
      tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
  webhookConfig:
    overrideScheduling:
      nodeSelector:
        node-role.kubernetes.io/control-plane: ''
      tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule
  cainjectorConfig:
    overrideScheduling:
      nodeSelector:
        node-role.kubernetes.io/control-plane: ''
      tolerations:
        - key: node-role.kubernetes.io/master
          operator: Exists
          effect: NoSchedule"
"
```

1. オーバーライド可能なスケジューリングパラメーターの詳細は、「CertManager カスタムリソースのフィールドの説明」の「cert-manager コンポーネントのオーバーライド可能なスケジューリングパラメーター」を参照してください。

検証

1. **cert-manager** Pod の Pod スケジューリング設定を検証します。
   a. 次のコマンドを実行して、**cert-manager** namespace のデプロイメントをチェックし、正しい **nodeSelector** と **tolerations** があることを確認します。

```
$ oc get pods -n cert-manager -o wide
```

出力例

```
NAME                                       READY   STATUS    RESTARTS   AGE   IP            NODE                         NOMINATED NODE   READINESS GATES
cert-manager-58d9c69db4-78mzp              1/1     Running   0          10m   10.129.0.36   ip-10-0-1-106.ec2.internal   <none>           <none>
cert-manager-cainjector-85b6987c66-rhzf7   1/1     Running   0          11m   10.128.0.39   ip-10-0-1-136.ec2.internal   <none>           <none>
cert-manager-webhook-7f54b4b858-29bsp      1/1     Running   0          11m   10.129.0.35   ip-10-0-1-106.ec2.internal   <none>           <none>
```

   b. 次のコマンドを実行して、デプロイメントに適用されている **nodeSelector** と **tolerations** の設定を確認します。

```
$ oc get deployments -n cert-manager -o jsonpath='{range .items[*]}{.metadata.name}{"\n"}{.spec.template.spec.nodeSelector}{"\n"}{.spec.template.spec.tolerations}{"\n\n"}{end}'
```

出力例

```
cert-manager
{"kubernetes.io/os":"linux","node-role.kubernetes.io/control-plane":""}
[{"effect":"NoSchedule","key":"node-role.kubernetes.io/master","operator":"Exists"}]

cert-manager-cainjector
{"kubernetes.io/os":"linux","node-role.kubernetes.io/control-plane":""}
[{"effect":"NoSchedule","key":"node-role.kubernetes.io/master","operator":"Exists"}]

cert-manager-webhook
{"kubernetes.io/os":"linux","node-role.kubernetes.io/control-plane":""}
[{"effect":"NoSchedule","key":"node-role.kubernetes.io/master","operator":"Exists"}]
```

2. 次のコマンドを実行して、**cert-manager** namespace 内の Pod スケジューリングイベントを検証します。

```
$ oc get events -n cert-manager --field-selector reason=Scheduled
```

関連情報

- CertManager カスタムリソースのフィールドの説明

9.6. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT の認証

クラウド認証情報を設定して、クラスター上で cert-manager Operator for Red Hat OpenShift を認証できます。

9.6.1. AWS での認証

前提条件

- cert-manager Operator for Red Hat OpenShift のバージョン 1.11.1 以降をインストールした。
- **mint** モードまたは **passthrough** モードで動作するように Cloud Credential Operator を設定した。

手順

1. 以下のように **CredentialsRequest** リソース YAML ファイル (例: **sample-credential-request.yaml**) を作成します。

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - "route53:GetChange"
      effect: Allow
      resource: "arn:aws:route53:::change/*"
    - action:
      - "route53:ChangeResourceRecordSets"
      - "route53:ListResourceRecordSets"
      effect: Allow
      resource: "arn:aws:route53:::hostedzone/*"
    - action:
      - "route53:ListHostedZonesByName"
      effect: Allow
      resource: "*"
  secretRef:
    name: aws-creds
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

2. 次のコマンドを実行して、**CredentialsRequest** リソースを作成します。

```
$ oc create -f sample-credential-request.yaml
```

3. 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のサブスクリプションオブジェクトを更新します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --type=merge -p '{"spec":{"config":{"env":[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"aws-creds"}]}}}'
```

検証

1. 次のコマンドを実行して、再デプロイされた cert-manager コントローラー Pod の名前を取得します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

出力例

```
NAME                          READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. 次のコマンドを実行して、cert-manager コントローラー Pod が、**mountPath** で指定されたパスの下にマウントされている AWS 認証情報ボリュームで更新されていることを確認します。

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

出力例

```
...
spec:
  containers:
  - args:
    ...
    - mountPath: /.aws
      name: cloud-credentials
  ...
  volumes:
  ...
  - name: cloud-credentials
    secret:
      ...
      secretName: aws-creds
```

9.6.2. AWS セキュリティートークンサービスによる認証

前提条件

- **ccoctl** バイナリーを抽出して準備している。
- Cloud Credential Operator を手動モードで使用し、AWS STS で OpenShift Container Platform クラスターを設定している。

手順

1. 次のコマンドを実行して、**CredentialsRequest** リソース YAML ファイルを保存するディレクトリーを作成します。

```
$ mkdir credentials-request
```

2. 次の yaml を適用して、**sample-credential-request.yaml** などの **CredentialsRequest** リソース YAML ファイルを **credentials-request** ディレクトリーの下に作成します。

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: AWSProviderSpec
    statementEntries:
    - action:
      - "route53:GetChange"
      effect: Allow
      resource: "arn:aws:route53:::change/*"
    - action:
      - "route53:ChangeResourceRecordSets"
      - "route53:ListResourceRecordSets"
      effect: Allow
      resource: "arn:aws:route53:::hostedzone/*"
    - action:
      - "route53:ListHostedZonesByName"
      effect: Allow
      resource: "*"
  secretRef:
    name: aws-creds
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

3. 次のコマンドを実行し、**ccoctl** ツールを使用して **CredentialsRequest** オブジェクトを処理します。

```
$ ccoctl aws create-iam-roles \
    --name <user_defined_name> --region=<aws_region> \
    --credentials-requests-dir=<path_to_credrequests_dir> \
    --identity-provider-arn <oidc_provider_arn> --output-dir=<path_to_output_dir>
```

出力例

```
2023/05/15 18:10:34 Role arn:aws:iam::XXXXXXXXXXXX:role/<user_defined_name>-cert-manager-aws-creds created
2023/05/15 18:10:34 Saved credentials configuration to: <path_to_output_dir>/manifests/cert-manager-aws-creds-credentials.yaml
2023/05/15 18:10:35 Updated Role policy for Role <user_defined_name>-cert-manager-aws-creds
```

次のステップで使用するために、出力から **<aws_role_arn>** をコピーします。例: **"arn:aws:iam::XXXXXXXXXXXX:role/<user_defined_name>-cert-manager-aws-creds"**

4. 次のコマンドを実行して、サービスアカウントに **eks.amazonaws.com/role-arn="<aws_role_arn>"** アノテーションを追加します。

```
$ oc -n cert-manager annotate serviceaccount cert-manager eks.amazonaws.com/role-arn="
<aws_role_arn>"
```

5. 新規 Pod を作成するには、以下のコマンドを実行して既存の cert-manager コントローラー Pod を削除します。

```
$ oc delete pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

AWS 認証情報は、1 分以内に新規 cert-manager コントローラー Pod に適用されます。

検証

1. 次のコマンドを実行して、更新された cert-manager コントローラー Pod の名前を取得します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

出力例

```
NAME                          READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          39s
```

2. 次のコマンドを実行して、AWS 認証情報が更新されていることを確認します。

```
$ oc set env -n cert-manager po/<cert_manager_controller_pod_name> --list
```

出力例

```
# pods/cert-manager-57f9555c54-vbcpg, container cert-manager-controller
# POD_NAMESPACE from field path metadata.namespace
AWS_ROLE_ARN=XXXXXXXXXXXX
AWS_WEB_IDENTITY_TOKEN_FILE=/var/run/secrets/eks.amazonaws.com/serviceaccount/to
ken
```

関連情報

- Cloud Credential Operator ユーティリティーの設定

9.6.3. Google Cloud での認証

前提条件

- cert-manager Operator for Red Hat OpenShift のバージョン 1.11.1 以降をインストールした。
- **mint** モードまたは **passthrough** モードで動作するように Cloud Credential Operator を設定した。

手順

1. 次の yaml を適用して、**sample-credential-request.yaml** などの **CredentialsRequest** リソース YAML ファイルを作成します。

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: GCPProviderSpec
    predefinedRoles:
    - roles/dns.admin
  secretRef:
    name: gcp-credentials
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

> **注記**
>
> **dns.admin** ロールは、Google Cloud DNS リソースを管理するための管理者権限をサービスアカウントに付与します。最小限の権限を持つサービスアカウントで cert-manager が実行されるようにするには、次の権限を持つカスタムロールを作成できます。
>
> - **dns.resourceRecordSets.***
> - **dns.changes.***
> - **dns.managedZones.list**

2. 次のコマンドを実行して、**CredentialsRequest** リソースを作成します。

```
$ oc create -f sample-credential-request.yaml
```

3. 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のサブスクリプションオブジェクトを更新します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --type=merge -p '{"spec":{"config":{"env":[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"gcp-credentials"}]}}}'
```

検証

1. 次のコマンドを実行して、再デプロイされた cert-manager コントローラー Pod の名前を取得します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

出力例

```
NAME                          READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. 次のコマンドを実行して、cert-manager コントローラー Pod が、**mountPath** で指定されたパスの下にマウントされている Google Cloud 認証情報ボリュームで更新されていることを確認します。

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

出力例

```
spec:
  containers:
  - args:
    ...
    volumeMounts:
    ...
    - mountPath: /.config/gcloud
      name: cloud-credentials
    ....
  volumes:
  ...
  - name: cloud-credentials
    secret:
      ...
      items:
      - key: service_account.json
        path: application_default_credentials.json
      secretName: gcp-credentials
```

9.6.4. Google Cloud Workload Identity による認証

前提条件

- **ccoctl** バイナリーを展開して準備した。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.11.1 以降をインストールした。
- Cloud Credential Operator を手動モードで使用して、Google Cloud Workload Identity で OpenShift Container Platform クラスターを設定した。

手順

1. 次のコマンドを実行して、**CredentialsRequest** リソース YAML ファイルを保存するディレクトリーを作成します。

```
$ mkdir credentials-request
```

2. **credentials-request** ディレクトリーに、次の **CredentialsRequest** マニフェストを含む YAML ファイルを作成します。

```
apiVersion: cloudcredential.openshift.io/v1
kind: CredentialsRequest
metadata:
  name: cert-manager
  namespace: openshift-cloud-credential-operator
spec:
  providerSpec:
    apiVersion: cloudcredential.openshift.io/v1
    kind: GCPProviderSpec
    predefinedRoles:
    - roles/dns.admin
  secretRef:
    name: gcp-credentials
    namespace: cert-manager
  serviceAccountNames:
  - cert-manager
```

注記

dns.admin ロールは、Google Cloud DNS リソースを管理するための管理者権限をサービスアカウントに付与します。最小限の権限を持つサービスアカウントで cert-manager が実行されるようにするには、次の権限を持つカスタムロールを作成できます。

- **dns.resourceRecordSets.***
- **dns.changes.***
- **dns.managedZones.list**

3. 次のコマンドを実行し、**ccoctl** ツールを使用して **CredentialsRequest** オブジェクトを処理します。

```
$ ccoctl gcp create-service-accounts \
    --name <user_defined_name> --output-dir=<path_to_output_dir> \
    --credentials-requests-dir=<path_to_credrequests_dir> \
    --workload-identity-pool <workload_identity_pool> \
    --workload-identity-provider <workload_identity_provider> \
    --project <gcp_project_id>
```

コマンドの例

```
$ ccoctl gcp create-service-accounts \
    --name abcde-20230525-4bac2781 --output-dir=/home/outputdir \
    --credentials-requests-dir=/home/credentials-requests \
    --workload-identity-pool abcde-20230525-4bac2781 \
    --workload-identity-provider abcde-20230525-4bac2781 \
    --project openshift-gcp-devel
```

4. 次のコマンドを実行して、クラスターのマニフェストディレクトリーに生成されたシークレットを適用します。

```
$ ls <path_to_output_dir>/manifests/*-credentials.yaml | xargs -I{} oc apply -f {}
```

5. 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のサブスクリプションオブジェクトを更新します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --type=merge -p '{"spec":{"config":{"env":[{"name":"CLOUD_CREDENTIALS_SECRET_NAME","value":"gcp-credentials"}]}}}'
```

検証

1. 次のコマンドを実行して、再デプロイされた cert-manager コントローラー Pod の名前を取得します。

```
$ oc get pods -l app.kubernetes.io/name=cert-manager -n cert-manager
```

出力例

```
NAME                          READY   STATUS    RESTARTS   AGE
cert-manager-bd7fbb9fc-wvbbt  1/1     Running   0          15m39s
```

2. 次のコマンドを実行して、cert-manager コントローラー Pod が、**mountPath** で指定されたパスの下にマウントされている Google Cloud Workload Identity 認証情報ボリュームで更新されていることを確認します。

```
$ oc get -n cert-manager pod/<cert-manager_controller_pod_name> -o yaml
```

出力例

```
spec:
  containers:
  - args:
    ...
    volumeMounts:
    - mountPath: /var/run/secrets/openshift/serviceaccount
      name: bound-sa-token
      ...
    - mountPath: /.config/gcloud
      name: cloud-credentials
  ...
  volumes:
  - name: bound-sa-token
    projected:
      ...
      sources:
      - serviceAccountToken:
          audience: openshift
          ...
          path: token
  - name: cloud-credentials
    secret:
      ...
      items:
      - key: service_account.json
        path: application_default_credentials.json
      secretName: gcp-credentials
```

■

関連情報

- Cloud Credential Operator ユーティリティーの設定
- コンポーネントの短期認証情報を使用した手動モード
- Cloud Credential Operator のデフォルト動作

9.7. ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift は、**Let's Encrypt** などの Automated Certificate Management Environment (ACME) CA サーバーを使用した証明書の発行をサポートしています。明示的な認証情報は、**Issuer** API オブジェクトにシークレットの詳細を指定して設定されます。アンビエント認証情報は、**Issuer** API オブジェクトで明示的に設定されていない環境、メタデータサービス、またはローカルファイルから抽出されます。

注記

Issuer オブジェクトのスコープは namespace です。同じ namespace からのみ証明書を発行できます。**ClusterIssuer** オブジェクトを使用して、クラスター内のすべての namespace で証明書を発行することもできます。

ClusterIssuer オブジェクトを定義する YAML ファイル例

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: acme-cluster-issuer
spec:
  acme:
    ...
```

注記

デフォルトで、**ClusterIssuer** オブジェクトをアンビエント認証情報と共に使用できます。**Issuer** オブジェクトをアンビエント認証情報で使用するには、cert-manager コントローラーの **--issuer-ambient-credentials** 設定を有効にする必要があります。

9.7.1. ACME 発行者について

cert-manager Operator for Red Hat OpenShift の ACME 発行者タイプは、Automated Certificate Management Environment (ACME) 認証局 (CA) サーバーを表します。ACME CA サーバーは、証明書が要求されているドメイン名をクライアントが所有していることを確認する **チャレンジ** に依存しています。チャレンジが成功すると、cert-manager Operator for Red Hat OpenShift が証明書を発行できます。チャレンジが失敗すると、cert-manager Operator for Red Hat OpenShift は証明書を発行しません。

注記

プライベート DNS ゾーンは、**Let's Encrypt** および internet ACME サーバーではサポートされません。

9.7.1.1. サポートされている ACME チャレンジタイプ

cert-manager Operator for Red Hat OpenShift は、ACME 発行者の次のチャレンジタイプをサポートします。

HTTP-01

HTTP-01 challenge タイプでは、ドメインの HTTP URL エンドポイントで計算されたキーを提供します。ACME CA サーバーが URL からキーを取得できる場合は、ドメインの所有者であることを確認できます。
詳細は、アップストリームの cert-manager ドキュメントの HTTP01 を参照してください。

> **注記**
>
> HTTP-01 では、Let's Encrypt サーバーがクラスターのルートにアクセスできる必要があります。内部クラスターまたはプライベートクラスターがプロキシーの背後にある場合、証明書発行の HTTP-01 検証は失敗します。
>
> HTTP-01 challenge はポート 80 に制限されています。詳細は、HTTP-01 challenge (Let's Encrypt) を参照してください。

DNS-01

DNS-01 チャレンジタイプでは、DNS TXT レコードで計算キーを提供します。ACME CA サーバーが DNS ルックアップによってキーを取得できる場合は、ドメインの所有者であることを確認できます。
詳細は、アップストリームの cert-manager ドキュメントの DNS01 を参照してください。

9.7.1.2. サポートされている DNS-01 プロバイダー

cert-manager Operator for Red Hat OpenShift は、ACME 発行者の次の DNS-01 プロバイダーをサポートします。

- Amazon Route 53
- Azure DNS

  > **注記**
  >
  > cert-manager Operator for Red Hat OpenShift は、Microsoft Entra ID Pod ID を使用して Pod にマネージドアイデンティティーを割り当てることをサポートしていません。

- Google Cloud DNS
- Webhook
  Red Hat は、cert-manager on OpenShift Container Platform と外部 webhook を使用して、DNS プロバイダーをテストおよびサポートします。次の DNS プロバイダーは OpenShift Container Platform でテストされ、サポートされています。
  - cert-manager-webhook-ibmcis

注記

リストされていない DNS プロバイダーを使用しても OpenShift Container Platform で動作する可能性はありますが、そのプロバイダーは Red Hat でテストされていないため、Red Hat サポートの対象には含まれません。

9.7.2. HTTP-01 チャレンジを解決するための ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して、ACME 発行者を設定し、HTTP-01 チャレンジを解決できます。この手順では、**Let's Encrypt** を ACME CA サーバーとして使用します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- 公開したいサービスがあります。この手順では、サービスの名前は **sample-workload** です。

手順

1. ACME クラスター発行者を作成します。

   a. **ClusterIssuer** オブジェクトを定義する YAML ファイルを作成します。

   acme-cluster-issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: ClusterIssuer
metadata:
  name: letsencrypt-staging                                    (1)
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_for_private_key>                           (2)
    server: https://acme-staging-v02.api.letsencrypt.org/directory (3)
    solvers:
    - http01:
        ingress:
          ingressClassName: openshift-default                  (4)
```

(1) クラスター発行者の名前を指定します。

(2) **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。

(3) ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。

(4) Ingress クラスを指定します。

   b. オプション: **ingressClassName** を指定せずにオブジェクトを作成する場合は、次のコマンドを使用して既存の Ingress にパッチを適用します。

```
$ oc patch ingress/<ingress-name> --type=merge --patch '{"spec":
{"ingressClassName":"openshift-default"}}' -n <namespace>
```

c. 次のコマンドを実行して、**ClusterIssuer** オブジェクトを作成します。

```
$ oc create -f acme-cluster-issuer.yaml
```

2. Ingress を作成して、ユーザーワークロードのサービスを公開します。

a. **Namespace** オブジェクトを定義する YAML ファイルを作成します。

namespace.yaml ファイルの例

```
apiVersion: v1
kind: Namespace
metadata:
  name: my-ingress-namespace 1
```

1 Ingress の namespace を指定します。

b. 次のコマンドを実行して、**Namespace** オブジェクトを作成します。

```
$ oc create -f namespace.yaml
```

c. **Ingress** オブジェクトを定義する YAML ファイルを作成します。

ingress.yaml ファイルの例

```
apiVersion: networking.k8s.io/v1
kind: Ingress
metadata:
  name: sample-ingress 1
  namespace: my-ingress-namespace 2
  annotations:
    cert-manager.io/cluster-issuer: letsencrypt-staging 3
spec:
  ingressClassName: openshift-default 4
  tls:
  - hosts:
    - <hostname> 5
    secretName: sample-tls 6
  rules:
  - host: <hostname> 7
    http:
      paths:
      - path: /
        pathType: Prefix
        backend:
          service:
```

```
          name: sample-workload
          port:
            number: 80
```

1. Ingress の名前を指定します。
2. Ingress 用に作成した namespace を指定します。
3. 作成したクラスター発行者を指定します。
4. Ingress クラスを指定します。
5. **<hostname>** は、証明書に関連付けるサブジェクト代替名 (SAN) に置き換えます。この名前は、DNS 名を証明書に追加するために使用されます。
6. 証明書を保存するシークレットを指定します。
7. **<hostname>** をホスト名に置き換えます。**<host_name>.<cluster_ingress_domain>** 構文を使用して、***.<cluster_ingress_domain>** ワイルドカード DNS レコードとクラスターのサービング証明書を利用できます。たとえば、**apps.<cluster_base_domain>** を使用できます。それ以外の場合は、選択したホスト名の DNS レコードが存在することを確認する必要があります。
8. 公開するサービスの名前を指定します。この例では、**sample-workload** という名前のサービスを使用しています。

d. 次のコマンドを実行して、**Ingress** オブジェクトを作成します。

```
$ oc create -f ingress.yaml
```

9.7.3. AWS Route53 の明示的な認証情報を使用した ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して自動証明書管理環境 (ACME) 発行者を設定し、AWS で明示的な認証情報を使用して DNS-01 チャレンジを解決できます。この手順では、**Let's Encrypt** を ACME 認証局 (CA) サーバーとして使用し、Amazon Route 53 を使用して DNS-01 チャレンジを解決する方法を示します。

前提条件

- 明示的な **accessKeyID** および **secretAccessKey** 認証情報を指定する。詳細は、アップストリームの cert-manager ドキュメントの Route53 を参照してください。

> **注記**
>
> AWS で実行されていない OpenShift Container Platform クラスターで明示的な認証情報を指定して Amazon Route 53 を使用できる。

手順

1. オプション: DNS-01 セルフチェックのネームサーバー設定をオーバーライドします。
   この手順は、ターゲットのパブリックホストゾーンがクラスターのデフォルトのプライベートホストゾーンに重複する場合のみ、必要です。

a. 次のコマンドを実行して、**CertManager** リソースを編集します。

```
$ oc edit certmanager cluster
```

b. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
metadata:
  name: cluster
  ...
spec:
  ...
  controllerConfig:                    (1)
    overrideArgs:
      - '--dns01-recursive-nameservers-only'    (2)
      - '--dns01-recursive-nameservers=1.1.1.1:53' (3)
```

1. **spec.controllerConfig** セクションを追加します。
2. そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。
3. DNS-01 セルフチェックをクエリーする **<host>:<port>** ネームサーバーのコンマ区切りリストを提供します。パブリックゾーンとプライベートゾーンの重複を避けるには、**1.1.1.1:53** 値を使用する必要があります。

c. 変更を適用するためにファイルを保存します。

2. オプション: 発行者の namespace を作成します。

```
$ oc new-project <issuer_namespace>
```

3. 次のコマンドを実行して、AWS 認証情報を保存するシークレットを作成します。

```
$ oc create secret generic aws-secret --from-literal=awsSecretAccessKey=
<aws_secret_access_key> \ (1)
    -n my-issuer-namespace
```

1. **<aws_secret_access_key>** を AWS シークレットアクセスキーに置き換えます。

4. 発行者を作成します。

a. **Issuer** オブジェクトを定義する YAML ファイルを作成します。

issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <letsencrypt_staging>          (1)
  namespace: <issuer_namespace>          (2)
```

```
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory 3
    email: "<email_address>" 4
    privateKeySecretRef:
      name: <secret_private_key> 5
    solvers:
    - dns01:
        route53:
          accessKeyID: <aws_key_id> 6
          hostedZoneID: <hosted_zone_id> 7
          region: <region_name> 8
          secretAccessKeySecretRef:
            name: "aws-secret" 9
            key: "awsSecretAccessKey" 10
```

1. 発行者の名前を指定します。
2. 発行者用に作成した namespace を指定します。
3. ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。
4. **<email_address>** を自分のメールアドレスに置き換えます。
5. **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。
6. **<aws_key_id>** を AWS キー ID に置き換えます。
7. **<hosted_zone_id>** をホストゾーン ID に置き換えます。
8. **<region_name>** は、AWS リージョン名に置き換えます。たとえば、**us-east-1** などです。
9. 作成したシークレットの名前を指定します。
10. 作成したシークレット内の、AWS シークレットアクセスキーを格納しているキーを指定します。

b. 次のコマンドを実行して、**Issuer** オブジェクトを作成します。

```
$ oc create -f issuer.yaml
```

9.7.4. AWS でアンビエント認証情報を使用した ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して、AWS 上のアンビエント認証情報を使用して DNS-01 チャレンジを解決する ACME 発行者を設定できます。この手順では、**Let's Encrypt** を ACME CA サーバーとして使用し、Amazon Route 53 を使用して DNS-01 チャレンジを解決する方法を示します。

前提条件

- クラスターが AWS Security Token Service (STS) を使用するように設定されている場合は、**AWS Security Token Service クラスターの cert-manager Operator for Red Hat OpenShift のクラウド認証情報の設定** セクションの手順を実行した。
- クラスターで AWS STS を使用しない場合は、**AWS での cert-manager Operator for Red Hat OpenShift クラウド認証情報の設定** セクションの手順を実行した。

手順

1. オプション: DNS-01 セルフチェックのネームサーバー設定をオーバーライドします。
この手順は、ターゲットのパブリックホストゾーンがクラスターのデフォルトのプライベートホストゾーンに重複する場合のみ、必要です。
    a. 次のコマンドを実行して、**CertManager** リソースを編集します。

    ```
    $ oc edit certmanager cluster
    ```

    b. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

    ```
    apiVersion: operator.openshift.io/v1alpha1
    kind: CertManager
    metadata:
      name: cluster
      ...
    spec:
      ...
      controllerConfig: (1)
        overrideArgs:
          - '--dns01-recursive-nameservers-only' (2)
          - '--dns01-recursive-nameservers=1.1.1.1:53' (3)
    ```

    (1) **spec.controllerConfig** セクションを追加します。

    (2) そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。

    (3) DNS-01 セルフチェックをクエリーする **<host>:<port>** ネームサーバーのコンマ区切りリストを提供します。パブリックゾーンとプライベートゾーンの重複を避けるには、**1.1.1.1:53** 値を使用する必要があります。

    c. 変更を適用するためにファイルを保存します。

2. オプション: 発行者の namespace を作成します。

    ```
    $ oc new-project <issuer_namespace>
    ```

3. **CertManager** リソースを変更して、**--issuer-ambient-credentials** 引数を追加します。

    ```
    $ oc patch certmanager/cluster \
      --type=merge \
      -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
    ```

4. 発行者を作成します。

a. **Issuer** オブジェクトを定義する YAML ファイルを作成します。

issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <letsencrypt_staging> 1
  namespace: <issuer_namespace> 2
spec:
  acme:
    server: https://acme-staging-v02.api.letsencrypt.org/directory 3
    email: "<email_address>" 4
    privateKeySecretRef:
      name: <secret_private_key> 5
    solvers:
    - dns01:
        route53:
          hostedZoneID: <hosted_zone_id> 6
          region: us-east-1
```

1. 発行者の名前を指定します。
2. 発行者用に作成した namespace を指定します。
3. ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。
4. **<email_address>** を自分のメールアドレスに置き換えます。
5. **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。
6. **<hosted_zone_id>** をホストゾーン ID に置き換えます。

b. 次のコマンドを実行して、**Issuer** オブジェクトを作成します。

```
$ oc create -f issuer.yaml
```

9.7.5. Google Cloud DNS の明示的な認証情報を使用した ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して ACME 発行者を設定し、Google Cloud で明示的な認証情報を使用して DNS-01 チャレンジを解決できます。この手順では、**Let's Encrypt** を ACME CA サーバーとして使用し、Google Cloud DNS を使用して DNS-01 チャレンジを解決する方法を示します。

前提条件

- Google Cloud DNS に必要なロールを持つ Google Cloud サービスアカウントを設定した。詳細は、アップストリームの cert-manager ドキュメントの Google Cloud DNS を参照してください。

注記

Google Cloud で実行されていない OpenShift Container Platform クラスターでは、明示的な認証情報を使用して Google Cloud DNS を使用できます。

手順

1. オプション: DNS-01 セルフチェックのネームサーバー設定をオーバーライドします。
この手順は、ターゲットのパブリックホストゾーンがクラスターのデフォルトのプライベートホストゾーンに重複する場合のみ、必要です。

   a. 次のコマンドを実行して、**CertManager** リソースを編集します。

   ```
   $ oc edit certmanager cluster
   ```

   b. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

   ```
   apiVersion: operator.openshift.io/v1alpha1
   kind: CertManager
   metadata:
     name: cluster
     ...
   spec:
     ...
     controllerConfig: (1)
       overrideArgs:
         - '--dns01-recursive-nameservers-only' (2)
         - '--dns01-recursive-nameservers=1.1.1.1:53' (3)
   ```

   (1) **spec.controllerConfig** セクションを追加します。

   (2) そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。

   (3) DNS-01 セルフチェックをクエリーする **<host>:<port>** ネームサーバーのコンマ区切りリストを提供します。パブリックゾーンとプライベートゾーンの重複を避けるには、**1.1.1.1:53** 値を使用する必要があります。

   c. 変更を適用するためにファイルを保存します。

2. オプション: 発行者の namespace を作成します。

   ```
   $ oc new-project my-issuer-namespace
   ```

3. 次のコマンドを実行して、Google Cloud 認証情報を保存するシークレットを作成します。

   ```
   $ oc create secret generic clouddns-dns01-solver-svc-acct --from-file=service_account.json=
   <path/to/gcp_service_account.json> -n my-issuer-namespace
   ```

4. 発行者を作成します。

   a. **Issuer** オブジェクトを定義する YAML ファイルを作成します。

issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme_dns01_clouddns_issuer> 1
  namespace: <issuer_namespace> 2
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> 3
    server: https://acme-staging-v02.api.letsencrypt.org/directory 4
    solvers:
    - dns01:
        cloudDNS:
          project: <project_id> 5
          serviceAccountSecretRef:
            name: clouddns-dns01-solver-svc-acct 6
            key: service_account.json 7
```

1. 発行者の名前を指定します。
2. **<issuer_namespace>** は、発行者の namespace に置き換えます。
3. **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。
4. ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。
5. **<project_id>** は、Cloud DNS ゾーンを含む Google Cloud プロジェクトの名前に置き換えます。
6. 作成したシークレットの名前を指定します。
7. 作成したシークレット内の、Google Cloud シークレットアクセスキーを格納しているキーを指定します。

b. 次のコマンドを実行して、**Issuer** オブジェクトを作成します。

```
$ oc create -f issuer.yaml
```

9.7.6. Google Cloud でのアンビエント認証情報を使用した ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して ACME 発行者を設定し、Google Cloud でアンビエント認証情報を使用して DNS-01 チャレンジを解決できます。この手順では、**Let's Encrypt** を ACME CA サーバーとして使用し、Google Cloud DNS を使用して DNS-01 チャレンジを解決する方法を示します。

前提条件

- クラスターが Google Cloud Workload Identity を使用するように設定されている場合は、**Google Cloud Workload Identity を使用した cert-manager Operator for Red Hat OpenShift の設定** セクションの手順を実行した。
- クラスターで Google Cloud Workload Identity を使用しない場合は、**Google Cloud での the cert-manager Operator for Red Hat OpenShift クラウド認証情報の設定** セクションの手順を実行した。

手順

1. オプション: DNS-01 セルフチェックのネームサーバー設定をオーバーライドします。
この手順は、ターゲットのパブリックホストゾーンがクラスターのデフォルトのプライベートホストゾーンに重複する場合のみ、必要です。
    a. 次のコマンドを実行して、**CertManager** リソースを編集します。

    ```
    $ oc edit certmanager cluster
    ```

    b. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

    ```
    apiVersion: operator.openshift.io/v1alpha1
    kind: CertManager
    metadata:
      name: cluster
      ...
    spec:
      ...
      controllerConfig:                       (1)
        overrideArgs:
          - '--dns01-recursive-nameservers-only'      (2)
          - '--dns01-recursive-nameservers=1.1.1.1:53' (3)
    ```

    (1) **spec.controllerConfig** セクションを追加します。

    (2) そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。

    (3) DNS-01 セルフチェックをクエリーする **<host>:<port>** ネームサーバーのコンマ区切りリストを提供します。パブリックゾーンとプライベートゾーンの重複を避けるには、**1.1.1.1:53** 値を使用する必要があります。

    c. 変更を適用するためにファイルを保存します。

2. オプション: 発行者の namespace を作成します。

    ```
    $ oc new-project <issuer_namespace>
    ```

3. **CertManager** リソースを変更して、**--issuer-ambient-credentials** 引数を追加します。

    ```
    $ oc patch certmanager/cluster \
      --type=merge \
      -p='{"spec":{"controllerConfig":{"overrideArgs":["--issuer-ambient-credentials"]}}}'
    ```

4. 発行者を作成します。

   a. **Issuer** オブジェクトを定義する YAML ファイルを作成します。

   issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme_dns01_clouddns_issuer> 1
  namespace: <issuer_namespace>
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> 2
    server: https://acme-staging-v02.api.letsencrypt.org/directory 3
    solvers:
    - dns01:
        cloudDNS:
          project: <gcp_project_id> 4
```

   1. 発行者の名前を指定します。
   2. **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。
   3. ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。
   4. **<gcp_project_id>** は、Cloud DNS ゾーンを含む Google Cloud プロジェクトの名前に置き換えます。

   b. 次のコマンドを実行して、**Issuer** オブジェクトを作成します。

```
$ oc create -f issuer.yaml
```

9.7.7. Microsoft Azure DNS の明示的な認証情報を使用した ACME 発行者の設定

cert-manager Operator for Red Hat OpenShift を使用して ACME 発行者を設定し、Microsoft Azure で明示的な認証情報を使用して DNS-01 チャレンジを解決できます。この手順では、**Let's Encrypt** を ACME CA サーバーとして使用し、Azure DNS で DNS-01 チャレンジを解決する方法を示します。

前提条件

- Azure DNS に必要なロールでサービスプリンシパルを設定している。詳細は、アップストリームの cert-manager ドキュメントの Azure DNS を参照してください。

> **注記**
>
> Microsoft Azure で実行されていない OpenShift Container Platform クラスターについては、以下の手順を実行できます。

手順

1. オプション: DNS-01 セルフチェックのネームサーバー設定をオーバーライドします。
この手順は、ターゲットのパブリックホストゾーンがクラスターのデフォルトのプライベートホストゾーンに重複する場合のみ、必要です。

    a. 次のコマンドを実行して、**CertManager** リソースを編集します。

    ```
    $ oc edit certmanager cluster
    ```

    b. 次のオーバーライド引数を指定して、**spec.controllerConfig** セクションを追加します。

    ```
    apiVersion: operator.openshift.io/v1alpha1
    kind: CertManager
    metadata:
      name: cluster
      ...
    spec:
      ...
      controllerConfig:                    1
        overrideArgs:
          - '--dns01-recursive-nameservers-only'      2
          - '--dns01-recursive-nameservers=1.1.1.1:53' 3
    ```

    1 **spec.controllerConfig** セクションを追加します。

    2 そのドメインに関連付けられた権限のあるネームサーバーをチェックする代わりに、再帰的なネームサーバーのみを使用するように指定します。

    3 DNS-01 セルフチェックをクエリーする **<host>:<port>** ネームサーバーのコンマ区切りリストを提供します。パブリックゾーンとプライベートゾーンの重複を避けるには、**1.1.1.1:53** 値を使用する必要があります。

    c. 変更を適用するためにファイルを保存します。

2. オプション: 発行者の namespace を作成します。

    ```
    $ oc new-project my-issuer-namespace
    ```

3. 次のコマンドを実行して、Azure 認証情報を保存するシークレットを作成します。

    ```
    $ oc create secret generic <secret_name> --from-literal=
    <azure_secret_access_key_name>=<azure_secret_access_key_value> \ 1 2 3
       -n my-issuer-namespace
    ```

    1 **<secret_name>** は、シークレット名に置き換えます。

    2 **<azure_secret_access_key_name>** は、Azure シークレットアクセスキーの名前に置き換えます。

    3 **<azure_secret_access_key_value>** は、Azure 秘密鍵に置き換えます。

4. 発行者を作成します。

   a. **Issuer** オブジェクトを定義する YAML ファイルを作成します。

      issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: <acme-dns01-azuredns-issuer> (1)
  namespace: <issuer_namespace> (2)
spec:
  acme:
    preferredChain: ""
    privateKeySecretRef:
      name: <secret_private_key> (3)
    server: https://acme-staging-v02.api.letsencrypt.org/directory (4)
    solvers:
    - dns01:
        azureDNS:
          clientID: <azure_client_id> (5)
          clientSecretSecretRef:
            name: <secret_name> (6)
            key: <azure_secret_access_key_name> (7)
          subscriptionID: <azure_subscription_id> (8)
          tenantID: <azure_tenant_id> (9)
          resourceGroupName: <azure_dns_zone_resource_group> (10)
          hostedZoneName: <azure_dns_zone> (11)
          environment: AzurePublicCloud
```

(1) 発行者の名前を指定します。

(2) **<issuer_namespace>** は、発行者の namespace に置き換えます。

(3) **<secret_private_key>** を ACME アカウントの秘密鍵を保存するシークレットの名前に置き換えます。

(4) ACME サーバーの **directory** エンドポイントにアクセスするための URL を指定します。この例では、**Let's Encrypt** のステージング環境を使用します。

(5) **<azure_client_id>** は Azure クライアント ID に置き換えます。

(6) **<secret_name>** はクライアントシークレットの名前に置き換えます。

(7) **<azure_secret_access_key_name>** は、クライアントシークレットキーの名前に置き換えます。

(8) **<azure_subscription_id>** は、Azure サブスクリプション ID に置き換えます。

(9) **<azure_tenant_id>** は Azure テナント ID に置き換えます。

(10) **<azure_dns_zone_resource_group>** は Azure DNS ゾーンリソースグループの名前に置き換えます。

11 **<azure_dns_zone>** は Azure DNS ゾーンの名前に置き換えます。

b. 次のコマンドを実行して、**Issuer** オブジェクトを作成します。

```
$ oc create -f issuer.yaml
```

9.7.8. 関連情報

- AWS Security Token Service クラスター用 Red Hat OpenShift cert-manager Operator のクラウド認証情報の設定
- AWS での the cert-manager Operator for Red Hat OpenShift クラウド認証情報の設定
- Google Cloud Workload Identity を使用した cert-manager Operator for Red Hat OpenShift のクラウド認証情報の設定
- Google Cloud での cert-manager Operator for Red Hat OpenShift クラウド認証情報の設定

9.8. 発行者による証明書の設定

cert-manager Operator for Red Hat OpenShift を使用すると、クラスター内のワークロードだけでなく、クラスターの外部で対話するコンポーネントの証明書を、更新や発行などのタスクを処理して管理できます。

9.8.1. ユーザーワークロードの証明書を作成する

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift がインストールされている。

手順

1. 発行者を作成します。詳細は、「関連情報」セクションの「発行者の設定」を参照してください。
2. 証明書を作成します。
    a. **Certificate** オブジェクトを定義する YAML ファイル (例: **certificate.yaml**) を作成します。

certificate.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: <tls_cert> 1
  namespace: <issuer_namespace> 2
spec:
  isCA: false
  commonName: '<common_name>' 3
  secretName: <secret_name> 4
```

```
dnsNames:
- "<domain_name>" 5
issuerRef:
  name: <issuer_name> 6
  kind: Issuer
```

1. 証明書の名前を指定します。
2. 発行者の namespace を指定します。
3. コモンネーム (CN) を指定します。
4. 作成する、証明書を含むシークレットの名前を指定します。
5. ドメイン名を指定します。
6. 発行者の名前を指定します。

b. 次のコマンドを実行して、**Certificate** オブジェクトを作成します。

```
$ oc create -f certificate.yaml
```

検証

- 次のコマンドを実行して、証明書が作成され、使用できる状態になっていることを確認します。

```
$ oc get certificate -w -n <issuer_namespace>
```

証明書のステータスが **Ready** になると、クラスター上のワークロードは、生成された証明書シークレットの使用を開始できます。

9.8.2. API サーバーの証明書を作成する

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.13.0 以降がインストールされている。

手順

1. 発行者を作成します。詳細は、「関連情報」セクションの「発行者の設定」を参照してください。
2. 証明書を作成します。
   a. **Certificate** オブジェクトを定義する YAML ファイル (例: **certificate.yaml**) を作成します。

certificate.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
```

```
kind: Certificate
metadata:
  name: <tls_cert> (1)
  namespace: openshift-config
spec:
  isCA: false
  commonName: "api.<cluster_base_domain>" (2)
  secretName: <secret_name> (3)
  dnsNames:
  - "api.<cluster_base_domain>" (4)
  issuerRef:
    name: <issuer_name> (5)
    kind: Issuer
```

1. 証明書の名前を指定します。
2. コモンネーム (CN) を指定します。
3. 作成する、証明書を含むシークレットの名前を指定します。
4. API サーバーの DNS 名を指定します。
5. 発行者の名前を指定します。

b. 次のコマンドを実行して、**Certificate** オブジェクトを作成します。

```
$ oc create -f certificate.yaml
```

3. API サーバーの名前が付いた証明書を追加します。詳細は、「関連情報」セクションの「API サーバーの名前が付いた証明書を追加する」セクションを参照してください。

> **注記**
>
> 証明書が更新されていることを確認するには、証明書を作成した後に **oc login** コマンドを再度実行します。

検証

- 次のコマンドを実行して、証明書が作成され、使用できる状態になっていることを確認します。

```
$ oc get certificate -w -n openshift-config
```

証明書のステータスが **Ready** になると、クラスター上の API サーバーは、生成された証明書シークレットの使用を開始できます。

9.8.3. Ingress コントローラーの証明書を作成する

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。

- cert-manager Operator for Red Hat OpenShift のバージョン 1.13.0 以降がインストールされている。

手順

1. 発行者を作成します。詳細は、「関連情報」セクションの「発行者の設定」を参照してください。

2. 証明書を作成します。

   a. **Certificate** オブジェクトを定義する YAML ファイル (例: **certificate.yaml**) を作成します。

   certificate.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: <tls_cert> 1
  namespace: openshift-ingress
spec:
  isCA: false
  commonName: "apps.<cluster_base_domain>" 2
  secretName: <secret_name> 3
  dnsNames:
  - "apps.<cluster_base_domain>" 4
  - "*.apps.<cluster_base_domain>" 5
  issuerRef:
    name: <issuer_name> 6
    kind: Issuer
```

   1 証明書の名前を指定します。

   2 コモンネーム (CN) を指定します。

   3 作成する、証明書を含むシークレットの名前を指定します。

   4 5 ingress の DNS 名を指定します。

   6 発行者の名前を指定します。

   b. 次のコマンドを実行して、**Certificate** オブジェクトを作成します。

```
$ oc create -f certificate.yaml
```

3. デフォルトの ingress 証明書を置き換えます。詳細は、「関連情報」セクションの「デフォルトの ingress 証明書を置き換える」セクションを参照してください。

検証

- 次のコマンドを実行して、証明書が作成され、使用できる状態になっていることを確認します。

```
$ oc get certificate -w -n openshift-ingress
```

証明書のステータスが **Ready** になると、クラスター上の Ingress コントローラーは、生成された証明書シークレットの使用を開始できます。

9.8.4. 関連情報

- 発行者の設定
  - サポート対象の発行者タイプ
  - ACME 発行者の設定
- API サーバーの名前が付いた証明書の追加
- デフォルトの Ingress 証明書の置き換え

9.9. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT によるルート保護

OpenShift Container Platform では、シークレット経由で TLS 証明書を参照するための設定可能なオプションを提供するために、ルート API が拡張されました。外部で管理される証明書を使用してルートを作成する テクノロジープレビュー機能を有効にすると、手動介入によるエラーを最小限に抑え、証明書管理プロセスを効率化し、OpenShift Container Platform ルーターが参照された証明書を迅速に提供できるようになります。

重要

cert-manager Operator for Red Hat OpenShift を使用してルートを保護する機能は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

9.9.1. クラスター内のルートを保護するための証明書の設定

次の手順は、cert-manager Operator for Red Hat OpenShift を **Let's Encrypt** の ACME HTTP-01 チャレンジタイプとともに使用して、OpenShift Container Platform クラスター内のルートリソースを保護するプロセスを示しています。

前提条件

- cert-manager Operator for Red Hat OpenShift のバージョン 1.14.0 以降がインストールされている。
- **RouteExternalCertificate** フィーチャーゲートを有効にした。
- **routes/custom-host** サブリソースに対する **create** および **update** 権限がある。
- 公開する **Service** リソースがある。

手順

1. 次のコマンドを実行して、HTTP-01 ソルバーを設定する **Issuer** を作成します。その他の ACME 発行者タイプについては、「ACME 発行者の設定」を参照してください。

Issuer.yaml ファイルの例

```
$ oc create -f - << EOF
apiVersion: cert-manager.io/v1
kind: Issuer
metadata:
  name: letsencrypt-acme
  namespace: <namespace> 1
spec:
  acme:
    server: https://acme-v02.api.letsencrypt.org/directory
    privateKeySecretRef:
      name: letsencrypt-acme-account-key
    solvers:
     - http01:
         ingress:
           ingressClassName: openshift-default
EOF
```

1 **Issuer** が配置されている namespace を指定します。ルートの namespace と同じである必要があります。

2. 次のコマンドを実行して、ルートの **Certificate** オブジェクトを作成します。**secretName** は、cert-manager が発行および管理する TLS シークレットを指定し、次の手順のルートでも参照されます。

```
$ oc create -f - << EOF
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: example-route-cert
  namespace: <namespace> 1
spec:
  commonName: <hostname> 2
  dnsNames:
    - <hostname> 3
  usages:
    - server auth
  issuerRef:
    kind: Issuer
    name: letsencrypt-acme
  secretName: <secret_name> 4
EOF
```

1 **Certificate** リソースが配置されている **namespace** を指定します。ルートの namespace と同じである必要があります。

2 ルートのホスト名を使用して、証明書の共通名を指定します。

3 ルートのホスト名を、証明書の DNS 名に追加します。

4 証明書が含まれるシークレットの名前を指定します。

3. 次のコマンドを使用して、参照先のシークレットを読み取る権限をルーターのサービスアカウントに付与するための **Role** を作成します。

```
$ oc create role secret-reader \
  --verb=get,list,watch \
  --resource=secrets \
  --resource-name=<secret_name> \ 1
  --namespace=<namespace> 2
```

1 アクセスを許可するシークレットの名前を指定します。これは、**Certificate** リソースで指定された **secretName** と一致する必要があります。

2 シークレットとルートの両方が配置されている namespace を指定します。

4. 次のコマンドを実行して、ルーターのサービスアカウントを、新しく作成した **Role** リソースにバインドする **RoleBinding** リソースを作成します。

```
$ oc create rolebinding secret-reader-binding \
  --role=secret-reader \
  --serviceaccount=openshift-ingress:router \
  --namespace=<namespace> 1
```

1 シークレットとルートの両方が配置されている namespace を指定します。

5. 次のコマンドを実行して、エッジ TLS 終端とカスタムホスト名を使用するサービスリソースのルートを作成します。ホスト名は、次のステップで **Certificate** リソースを作成するときに使用します。

```
$ oc create route edge <route_name> \ 1
  --service=<service_name> \ 2
  --hostname=<hostname> \ 3
  --namespace=<namespace> 4
```

1 ルートの名前を指定します。

2 公開するサービスを指定します。

3 ルートのホスト名を指定します。

4 ルートが配置されている namespace を指定します。

6. 次のコマンドを使用して、ルートの **.spec.tls.externalCertificate** フィールドを更新し、以前に作成したシークレットを参照し、cert-manager が発行した証明書を使用します。

```
$ oc patch route <route_name> \ 1
  -n <namespace> \ 2
  --type=merge \
  -p '{"spec":{"tls":{"externalCertificate":{"name":"<secret_name>"}}}}' 3
```

1 ルート名を指定します。

2 シークレットとルートの両方が配置されている namespace を指定します。

3 証明書が含まれるシークレットの名前を指定します。

検証

- 次のコマンドを実行して、証明書が作成され、使用できる状態になっていることを確認します。

```
$ oc get certificate -n <namespace> 1
$ oc get secret -n <namespace> 2
```

1 2 シークレットとルートの両方が存在する namespace を指定します。

- 次のコマンドを実行して、ルーターが参照された外部証明書を使用していることを確認します。コマンドは、ステータスコード **200 OK** を返すはずです。

```
$ curl -IsS https://<hostname> 1
```

1 ルートのホスト名を指定します。

- 次のコマンドを実行して、サーバー証明書の **subject**、**subjectAltName**、**issuer** が、すべて curl verbose 出力から期待されるとおりであることを確認します。

```
$ curl -v https://<hostname> 1
```

1 ルートのホスト名を指定します。

これでルートは、cert-manager が発行した参照シークレットの証明書で正常に保護されました。cert-manager は証明書のライフサイクルを自動的に管理します。

9.9.2. 関連情報

- 外部管理証明書を使用したルートの作成
- ACME 発行者の設定

9.10. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT と ISTIO-CSR の統合

重要

cert-manager Operator for Red Hat OpenShift と Istio-CSR の統合は、テクノロジープレビュー機能です。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) の対象外であり、機能的に完全ではないことがあります。Red Hat は、実稼働環境でこれらを使用することを推奨していません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供して、開発段階で機能のテストを行い、フィードバックを提供していただくことを目的としています。

Red Hat のテクノロジープレビュー機能のサポート範囲に関する詳細は、テクノロジープレビュー機能のサポート範囲 を参照してください。

cert-manager Operator for Red Hat OpenShift は、Red Hat OpenShift Service Mesh または Istio のワークロードとコントロールプレーンコンポーネントを保護するための強化されたサポートを提供します。これには、相互 TLS (mTLS) を有効にする証明書のサポートが含まれます。この証明書は cert-manager の発行者を使用して署名、配信、更新されます。cert-manager Operator for Red Hat OpenShift で管理される Istio-CSR エージェントを使用して、Istio ワークロードとコントロールプレーンコンポーネントを保護できます。

この Istio-CSR 統合により、Istio は cert-manager Operator for Red Hat OpenShift から証明書を取得できるようになります。これにより、セキュリティーと証明書管理が簡素化されます。

9.10.1. cert-manager Operator for Red Hat OpenShift を介した Istio-CSR エージェントのインストール

9.10.1.1. Istio-CSR 機能の有効化

cert-manager Operator for Red Hat OpenShift で Istio-CSR 機能を有効にするには、この手順を使用します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

- 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のデプロイメントを更新し、config map を使用します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --type='merge' -p '{"spec":{"config":{"env":[{"name":"UNSUPPORTED_ADDON_FEATURES","value":"IstioCSR=true"}]}}}'
```

検証

1. 次のコマンドを実行して、デプロイメントのロールアウトが完了したことを確認します。

```
$ oc rollout status deployment/cert-manager-operator-controller-manager -n cert-manager-operator
```

出力例

```
deployment "cert-manager-operator-controller-manager" successfully rolled out
```

9.10.1.2. Istio-CSR エージェントのルート CA 発行者の作成

Istio-CSR エージェントのルート CA 発行者を作成するには、この手順を使用します。

注記

サポートされている他の発行者も使用できます。ただし、ACME はサポートされていません。詳細は、「cert-manager Operator for Red Hat OpenShift の発行者プロバイダー」を参照してください。

手順

1. **Issuer** および **Certificate** オブジェクトを定義する YAML ファイルを作成します。

issuer.yaml ファイルの例

```
apiVersion: cert-manager.io/v1
kind: Issuer (1)
metadata:
  name: selfsigned
  namespace: <istio_project_name> (2)
spec:
  selfSigned: {}
---
apiVersion: cert-manager.io/v1
kind: Certificate
metadata:
  name: istio-ca
  namespace: <istio_project_name>
spec:
  isCA: true
  duration: 87600h # 10 years
  secretName: istio-ca
  commonName: istio-ca
  privateKey:
    algorithm: ECDSA
    size: 256
  subject:
    organizations:
      - cluster.local
      - cert-manager
  issuerRef:
    name: selfsigned
    kind: Issuer (3)
    group: cert-manager.io
---
apiVersion: cert-manager.io/v1
kind: Issuer (4)
metadata:
  name: istio-ca
  namespace: <istio_project_name> (5)
spec:
  ca:
    secretName: istio-ca
```

1 3 4 **Issuer** または **ClusterIssuer** を指定します。

2 5 Istio プロジェクトの名前を指定します。

検証

- 次のコマンドを実行して、発行者が作成され、使用できる状態になっていることを確認します。

```
$ oc get issuer istio-ca -n <istio_project_name>
```

出力例

```
NAME       READY   AGE
istio-ca   True    3m
```

関連情報

- cert-manager Operator for Red Hat OpenShift 発行者プロバイダー

9.10.1.3. IstioCSR カスタムリソースの作成

cert-manager Operator for Red Hat OpenShift を介して Istio-CSR エージェントをインストールするには、この手順を使用します。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- Istio-CSR 機能を有効にした。
- Istio-CSR エージェントの証明書を生成するために必要な **Issuer** または **ClusterIssuer** リソースを作成した。

> **注記**
>
> **Issuer** リソースを使用している場合は、Red Hat OpenShift Service Mesh または **Istiod** namespace に **Issuer** および **Certificate** リソースを作成してください。証明書要求が同じ namespace に生成され、それに応じてロールベースのアクセス制御 (RBAC) が設定されます。

手順

1. 次のコマンドを実行して、Istio-CSR をインストールするための新しいプロジェクトを作成します。Istio-CSR をインストールするための既存のプロジェクトがある場合は、この手順をスキップしてください。

```
$ oc new-project <istio_csr_project_name>
```

2. **IstioCSR** カスタムリソースを作成して、cert-manager Operator for Red Hat OpenShift によって管理される Istio-CSR エージェントを有効にし、Istio ワークロードとコントロールプレーンの証明書署名要求を処理できるようにします。

注記

一度にサポートされる **IstioCSR** カスタムリソース (CR) は 1 つだけです。複数の **IstioCSR** CR が作成された場合、アクティブになるのは 1 つだけです。**IstioCSR** の **status** サブリソースを使用して、リソースが未処理かどうかを確認してください。

- 複数の **IstioCSR** CR が同時に作成された場合、いずれも処理されません。
- 複数の **IstioCSR** CR が連続して作成された場合、最初の CR のみが処理されます。
- 新しい要求が拒否されないように、未処理の **IstioCSR** CR を削除します。
- **IstioCSR** 用に作成されたオブジェクトが、Operator によって自動的に削除されることはありません。アクティブな **IstioCSR** リソースが削除され、以前のデプロイメントを削除せずに別の namespace に新しいリソースが作成されると、複数の **istio-csr** デプロイメントがアクティブなままになる可能性があります。この動作は推奨されておらず、サポートされていません。

a. **IstioCSR** オブジェクトを定義する YAML ファイルを作成します。

IstioCSR CR の例

```
apiVersion: operator.openshift.io/v1alpha1
kind: IstioCSR
metadata:
  name: default
  namespace: <istio_csr_project_name>
spec:
  istioCSRConfig:
    certManager:
      issuerRef:
        name: istio-ca 1
        kind: Issuer 2
        group: cert-manager.io
    istiodTLSConfig:
      trustDomain: cluster.local
    istio:
      namespace: <istio_project_name>
```

1 **Issuer** または **ClusterIssuer** の名前を指定します。**issuer.yaml** ファイルで定義されている CA 発行者と同じ名前にする必要があります。

2 **Issuer** または **ClusterIssuer** の種類を指定します。**issuer.yaml** ファイルで定義されている CA 発行者と同じ種類である必要があります。

b. 次のコマンドを実行して、**IstioCSR** カスタムリソースを作成します。

```
$ oc create -f IstioCSR.yaml
```

検証

1. 次のコマンドを実行して、Istio-CSR デプロイメントの準備ができていることを確認します。

```
$ oc get deployment -n <istio_csr_project_name>
```

出力例

```
NAME                    READY   UP-TO-DATE   AVAILABLE   AGE
cert-manager-istio-csr   1/1     1            1           24s
```

2. 次のコマンドを実行して、Istio-CSR の Pod が実行されていることを確認します。

```
$ oc get pod -n <istio_csr_project_name>
```

出力例

```
NAME                                      READY   STATUS    RESTARTS    AGE
cert-manager-istio-csr-5c979f9b7c-bv57w  1/1     Running  0           45s
```

- 次のコマンドを実行して、Istio-CSR の Pod がログにエラーを報告していないことを確認します。

```
$ oc -n <istio_csr_project_name> logs <istio_csr_pod_name>
```

- 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift の Pod がエラーを報告していないことを確認します。

```
$ oc -n cert-manager-operator logs <cert_manager_operator_pod_name>
```

9.10.2. cert-manager Operator for Red Hat OpenShift によって管理される Istio-CSR エージェントのアンインストール

Red Hat OpenShift の cert-manager Operator によって管理される Istio-CSR エージェントをアンインストールするには、この手順を使用します。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- Istio-CSR 機能を有効にした。
- **IstioCSR** カスタムリソースを作成した。

手順

1. 次のコマンドを実行して、**IstioCSR** カスタムリソースを削除します。

```
$ oc -n <istio_csr_project_name> delete istiocsrs.operator.openshift.io default
```

2. 関連リソースを削除します。

重要

Red Hat OpenShift Service Mesh または Istio コンポーネントの停止を回避するために、次のリソースを削除する前に、Istio-CSR サービスまたは Istio 用に発行された証明書を参照しているコンポーネントがないことを確認してください。

a. 次のコマンドを実行してクラスタースコープのリソースをリスト表示し、リスト表示されたリソースの名前を後で参照できるように保存します。

```
$ oc get clusterrolebindings,clusterroles -l "app=cert-manager-istio-csr,app.kubernetes.io/name=cert-manager-istio-csr"
```

b. 次のコマンドを実行して、Istio-csr がデプロイされた namespace 内のリソースをリスト表示し、リスト表示されたリソースの名前を後で参照できるように保存します。

```
$ oc get certificate,deployments,services,serviceaccounts -l "app=cert-manager-istio-csr,app.kubernetes.io/name=cert-manager-istio-csr" -n <istio_csr_project_name>
```

c. 次のコマンドを実行して、Red Hat OpenShift Service Mesh または Istio がデプロイされた namespace 内のリソースをリスト表示し、リスト表示されたリソースの名前を後で参照できるように保存します。

```
$ oc get roles,rolebindings -l "app=cert-manager-istio-csr,app.kubernetes.io/name=cert-manager-istio-csr" -n <istio_csr_project_name>
```

d. 前のステップでリスト表示された各リソースに対して、次のコマンドを実行してリソースを削除します。

```
$ oc -n <istio_csr_project_name> delete <resource_type>/<resource_name>
```

関連するリソースがすべて削除されるまで、このプロセスを繰り返します。

9.10.3. Istio-CSR 機能が有効な cert-manager Operator for Red Hat OpenShift のアップグレード

Istio-CSR TechPreview フィーチャーゲートが有効になっている場合、Operator をアップグレードすることはできません。次の利用可能なバージョンを使用するには、cert-manager Operator for Red Hat OpenShift をアンインストールし、すべての Istio-CSR リソースを削除してから再インストールする必要があります。

9.11. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT のモニタリング

デフォルトでは、cert-manager Operator for Red Hat OpenShift は、3 つのコアコンポーネントである controller、cainjector、および webhook のメトリクスを公開します。Prometheus Operator 形式を使用してこれらのメトリクスを収集するように OpenShift Monitoring を設定できます。

9.11.1. ユーザーワークロードモニタリングの有効化

クラスターでユーザーワークロードモニタリングを設定することで、ユーザー定義プロジェクトのモニタリングを有効にできます。詳細は、「ユーザー定義プロジェクトのメトリクス収集の設定」を参照してください。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **cluster-monitoring-config.yaml** YAML ファイルを作成します。

```
apiVersion: v1
kind: ConfigMap
metadata:
  name: cluster-monitoring-config
  namespace: openshift-monitoring
data:
  config.yaml: |
    enableUserWorkload: true
```

2. 次のコマンドを実行して **ConfigMap** を適用します。

```
$ oc apply -f cluster-monitoring-config.yaml
```

検証

1. 次のコマンドを実行して、ユーザーワークロードのモニタリングコンポーネントが **openshift-user-workload-monitoring** namespace で実行されていることを確認します。

```
$ oc -n openshift-user-workload-monitoring get pod
```

出力例

```
NAME                                   READY   STATUS    RESTARTS   AGE
prometheus-operator-6cb6bd9588-dtzxq   2/2     Running   0          50s
prometheus-user-workload-0             6/6     Running   0          48s
prometheus-user-workload-1             6/6     Running   0          48s
thanos-ruler-user-workload-0           4/4     Running   0          42s
thanos-ruler-user-workload-1           4/4     Running   0          42s
```

prometheus-operator、**prometheus-user-workload**、**thanos-ruler-user-workload** などの Pod のステータスが **Running** である必要があります。

関連情報

- ユーザー定義プロジェクトのメトリクス収集の設定

9.11.2. ServiceMonitor を使用した cert-manager Operator for Red Hat OpenShift オペランドのメトリクス収集の設定

cert-manager Operator for Red Hat OpenShift のオペランドは、デフォルトで **/metrics** サービスエンドポイントのポート **9402** でメトリクスを公開します。Prometheus Operator によるカスタムメトリクスの収集を可能にする **ServiceMonitor** カスタムリソース (CR) を作成することで、cert-manager オペランドのメトリクス収集を設定できます。詳細は、「ユーザーワークロードモニタリングの設定」を参照してください。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift がインストールされている。
- ユーザーワークロードモニタリングが有効になっている。

手順

1. **ServiceMonitor** CR を作成します。
    a. **ServiceMonitor** CR を定義する YAML ファイルを作成します。

    servicemonitor-cert-manager.yaml ファイルの例

```yaml
apiVersion: monitoring.coreos.com/v1
kind: ServiceMonitor
metadata:
  labels:
    app: cert-manager
    app.kubernetes.io/instance: cert-manager
    app.kubernetes.io/name: cert-manager
  name: cert-manager
  namespace: cert-manager
spec:
  endpoints:
    - honorLabels: false
      interval: 60s
      path: /metrics
      scrapeTimeout: 30s
      targetPort: 9402
  selector:
    matchExpressions:
      - key: app.kubernetes.io/name
        operator: In
        values:
          - cainjector
          - cert-manager
          - webhook
      - key: app.kubernetes.io/instance
        operator: In
        values:
          - cert-manager
      - key: app.kubernetes.io/component
        operator: In
        values:
          - cainjector
          - controller
          - webhook
```

    b. 次のコマンドを実行して **ServiceMonitor** CR を作成します。

```
$ oc apply -f servicemonitor-cert-manager.yaml
```

ServiceMonitor CR が作成されると、ユーザーワークロードの Prometheus インスタンスが、cert-manager Operator for Red Hat OpenShift オペランドからのメトリクス収集を開始します。収集されたメトリクスには、**job="cert-manager"**、**job="cert-manager-cainjector"**、**job="cert-manager-webhook"** というラベルが付けられます。

検証

1. OpenShift Container Platform Web コンソールで、**Observe** → **Targets** に移動します。

2. **Label** フィルターフィールドに次のラベルを入力して、各オペランドのメトリクスターゲットをフィルタリングします。

```
$ service=cert-manager
```

```
$ service=cert-manager-webhook
```

```
$ service=cert-manager-cainjector
```

3. **cert-manager**、**cert-manager-webhook**、**cert-manager-cainjector** エントリーの **Status** 列に **Up** と表示されていることを確認します。

関連情報

- ユーザーワークロードモニタリングの設定

9.11.3. cert-manager Operator for Red Hat OpenShift オペランドのメトリクスの照会

クラスター管理者、またはすべての namespace に対する表示アクセス権を持つユーザーは、OpenShift Container Platform Web コンソールまたはコマンドラインインターフェイス (CLI) を使用して、cert-manager Operator for Red Hat OpenShift のオペランドのメトリクスを照会できます。詳細は、「メトリクスへのアクセス」を参照してください。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift がインストールされている。
- **ServiceMonitor** オブジェクトを作成することで、モニタリングとメトリクスの収集を有効にした。

手順

1. OpenShift Container Platform Web コンソールで、**Observe** → **Metrics** に移動します。

2. クエリーフィールドに次の PromQL 式を入力して、各オペランドの cert-manager Operator for Red Hat OpenShift オペランドメトリクスを照会します。

```
{job="cert-manager"}
```

```
{job="cert-manager-webhook"}
```

```
{job="cert-manager-cainjector"}
```

関連情報

- メトリクスへのアクセス

9.12. CERT-MANAGER および CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT のログレベルの設定

cert-manager コンポーネントと cert-manager Operator for Red Hat OpenShift に関する問題をトラブルシューティングするには、ログの詳細レベルを設定できます。

注記

それぞれの cert-manager コンポーネントに異なるログレベルを使用する場合は、**cert-manager Operator API フィールドのカスタマイズ** を参照してください。

9.12.1. cert-manager のログレベル設定

cert-manager のログレベルを設定して、ログメッセージの詳細レベルを決定できます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.11.1 以降をインストールした。

手順

1. 次のコマンドを実行して、**CertManager** リソースを編集します。

```
$ oc edit certmanager.operator cluster
```

2. **spec.logLevel** セクションを編集してログレベルの値を設定します。

```
apiVersion: operator.openshift.io/v1alpha1
kind: CertManager
...
spec:
  logLevel: <log_level> 1
```

1 **CertManager** リソースの有効なログレベル値は **Normal**、**Debug**、**Trace**、および **TraceAll** です。ログを監査し、異常がない場合に一般的な操作を実行するには、**logLevel** を **Normal** に設定します。詳細なログを表示してマイナーな問題のトラブルシューティングを行う場合は、**logLevel** を **Debug** に設定します。さらに詳細なログを表示して重大な問題のトラブルシューティングを行う場合は、**logLevel** を **Trace** に設定します。深刻な問題のトラブルシューティングを行う場合は、**logLevel** を **TraceAll** に設定します。デフォルトの **logLevel** は **Normal** です。

> **注記**
>
> **TraceAll** は大量のログを生成します。**logLevel** を **TraceAll** に設定すると、パフォーマンスの問題が発生する可能性があります。

3. 変更を保存してテキストエディターを終了し、変更を適用します。
変更を適用すると、cert-manager コンポーネントコントローラー、CA インジェクター、Webhook の詳細レベルが更新されます。

9.12.2. cert-manager Operator for Red Hat OpenShift のログレベル設定

cert-manager Operator for Red Hat OpenShift のログレベルを設定して、Operator ログメッセージの詳細レベルを決定できます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- cert-manager Operator for Red Hat OpenShift のバージョン 1.11.1 以降をインストールした。

手順

- 次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift サブスクリプションオブジェクトを更新し、Operator ログの詳細レベルを指定します。

```
$ oc -n cert-manager-operator patch subscription openshift-cert-manager-operator --type='merge' -p '{"spec":{"config":{"env":[{"name":"OPERATOR_LOG_LEVEL","value":"v"}]}}}'
1
```

1. **v** を任意のログレベル番号に置き換えます。**v** の有効な値の範囲は **1`to `10** です。デフォルト値は **2** です。

検証

1. cert-manager Operator Pod が再デプロイされます。次のコマンドを実行して、cert-manager Operator for Red Hat OpenShift のログレベルが更新されているか確認します。

```
$ oc set env deploy/cert-manager-operator-controller-manager -n cert-manager-operator --list | grep -e OPERATOR_LOG_LEVEL -e container
```

出力例

```
# deployments/cert-manager-operator-controller-manager, container kube-rbac-proxy
OPERATOR_LOG_LEVEL=9
# deployments/cert-manager-operator-controller-manager, container cert-manager-operator
OPERATOR_LOG_LEVEL=9
```

2. **oc logs** コマンドを実行して、cert-manager Operator for Red Hat OpenShift のログレベルが更新されているか確認します。

```
$ oc logs deploy/cert-manager-operator-controller-manager -n cert-manager-operator
```

9.12.3. 関連情報

- cert-manager Operator API フィールドのカスタマイズ

9.13. CERT-MANAGER OPERATOR FOR RED HAT OPENSHIFT のアンインストール

OpenShift Container Platform から cert-manager Operator for Red Hat OpenShift を削除するには、Operator をアンインストールし、関連リソースを削除します。

9.13.1. cert-manager Operator for Red Hat OpenShift のアンインストール

Web コンソールを使用して、cert-manager Operator for Red Hat OpenShift をアンインストールできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。
- cert-manager Operator for Red Hat OpenShift がインストールされている。

手順

1. OpenShift Container Platform Web コンソールにログインします。
2. cert-manager Operator for Red Hat OpenShift をアンインストールします。
   a. **Operators → Installed Operators** に移動します。
   b. **cert-manager Operator for Red Hat OpenShift**エントリーの横にある Options メニュー ⋮ をクリックし、**Uninstall Operator** をクリックします。
   c. 確認ダイアログで、**Uninstall** をクリックします。

9.13.2. cert-manager Operator for Red Hat OpenShift リソースの削除

cert-manager Operator for Red Hat OpenShift をアンインストールしたら、その関連リソースをクラスターから削除することもできます。

前提条件

- **cluster-admin** 権限でクラスターにアクセスできる。
- OpenShift Container Platform Web コンソールにアクセスできる。

手順

1. OpenShift Container Platform Web コンソールにログインします。

2. **cert-manager** namespace にある **cert-manager**、**cainjector**、**webhook** などの cert-manager コンポーネントのデプロイメントを削除します。
   a. **Project** ドロップダウンメニューをクリックして利用可能なプロジェクトのリストを表示し、**cert-manager** プロジェクトを選択します。
   b. **Workloads** → **Deployments** に移動します。
   c. 削除するデプロイメントを選択します。
   d. **Actions** ドロップダウンメニューをクリックし、**Delete Deployment** を選択して確認ダイアログボックスを表示します。
   e. **Delete** をクリックしてデプロイメントを削除します。
   f. または、コマンドラインインターフェイス (CLI) を使用して、**cert-manager** namespace に存在する **cert-manager**、**cainjector**、**webhook** などの cert-manager コンポーネントのデプロイメントを削除します。

```
$ oc delete deployment -n cert-manager -l app.kubernetes.io/instance=cert-manager
```

3. オプション: cert-manager Operator for Red Hat OpenShift によってインストールされたカスタムリソース定義 (CRD) を削除します。
   a. 次のコマンドを実行して、**CertManager** カスタムリソース(CR)からファイナライザーを削除します。

```
$ oc patch certmanagers.operator cluster --type=merge -p='{"metadata":
{"finalizers":null}}'
```

   b. **Administration** → **CustomResourceDefinitions** に移動します。
   c. **Name** フィールドに **certmanager** と入力して、CRD をフィルタリングします。
   d. 次の各 CRD の横にある Options メニュー ⋮ をクリックし、**Delete Custom Resource Definition** を選択します。
      - **Certificate**
      - **CertificateRequest**
      - **CertManager** (**operator.openshift.io**)
      - **Challenge**
      - **ClusterIssuer**
      - **Issuer**
      - **Order**
4. オプション: **cert-manager-operator** namespace を削除します。
   a. **Administration** → **Namespaces** に移動します。

b. **cert-manager-operator** の横にある Options オプションメニュー ⋮ をクリックし、**Delete Namespace** を選択します。

c. 確認ダイアログで、フィールドに **cert-manager-operator** と入力し、**Delete** をクリックします。

第10章 監査ログの表示

OpenShift Container Platform 監査は、システムに影響を与えた一連のアクティビティーを個別のユーザー、管理者その他システムのコンポーネント別に記述したセキュリティー関連の時系列のレコードを提供します。

10.1. API の監査ログについて

監査は API サーバーレベルで実行され、サーバーに送られるすべての要求をログに記録します。それぞれの監査ログには、以下の情報が含まれます。

表10.1 監査ログフィールド

フィールド	説明
level	イベントが生成された監査レベル。
auditID	要求ごとに生成される一意の監査 ID。
stage	このイベントインスタンスの生成時の要求処理のステージ。
requestURI	クライアントによってサーバーに送信される要求 URI。
verb	要求に関連付けられる Kubernetes の動詞。リソース以外の要求の場合、これは小文字の HTTP メソッドになります。
user	認証されたユーザーの情報。
impersonatedUser	オプション: 偽装ユーザーの情報 (要求で別のユーザーを偽装する場合)。
sourceIPs	オプション: 要求の送信元および中間プロキシーからのソース IP。
userAgent	オプション: クライアントが報告するユーザーエージェントの文字列。ユーザーエージェントはクライアントによって提供されており、信頼できないことに注意してください。
objectRef	オプション: この要求のターゲットとなっているオブジェクト参照。これは、**List** タイプの要求やリソース以外の要求には適用されません。
responseStatus	オプション: **ResponseObject** が **Status** タイプでなくても設定される応答ステータス。正常な応答の場合、これにはコードのみが含まれます。ステータス以外のタイプのエラー応答の場合、これにはエラーメッセージが自動的に設定されます。

フィールド	説明
requestObject	オプション: JSON 形式の要求からの API オブジェクト。**RequestObject** は、バージョンの変換、デフォルト値の設定、アドミッション、またはマージが行われる前に、(JSON 形式で再エンコードされる可能性もありますが) 要求に含まれるそのままの状態で記録されます。これは外部のバージョン付けされたオブジェクトタイプであり、それ自体では有効なオブジェクトではない可能性があります。これはリソース以外の要求の場合には省略され、要求レベル以上でのみログに記録されます。
responseObject	オプション: JSON 形式の応答で返される API オブジェクト。**ResponseObject** は外部タイプへの変換後に記録され、JSON としてシリアライズされます。これはリソース以外の要求の場合には省略され、応答レベルでのみログに記録されます。
requestReceivedTimestamp	要求が API サーバーに到達した時間。
stageTimestamp	要求が現在の監査ステージに達した時間。
annotations	オプション: 監査イベントと共に保存される構造化されていないキーと値のマップ。これは、認証、認可、受付プラグインなど、要求提供チェーンで呼び出されるプラグインによって設定される可能性があります。これらのアノテーションは監査イベント用のもので、送信されたオブジェクトの **metadata.annotations** に対応しないことに注意してください。キーは、名前の競合が発生しないように通知コンポーネントを一意に識別する必要があります (例: **podsecuritypolicy.admission.k8s.io/policy**)。値は短くする必要があります。アノテーションはメタデータレベルに含まれます。

Kubernetes API サーバーの出力例:


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"ad209ce1-fec7-4130-8192-c4cc63f1d8cd","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-kube-controller-manager/configmaps/cert-recovery-controller-lock?timeout=35s","verb":"update","user":{"username":"system:serviceaccount:openshift-kube-controller-manager:localhost-recovery-client","uid":"dd4997e3-d565-4e37-80f8-7fc122ccd785","groups":["system:serviceaccounts","system:serviceaccounts:openshift-kube-controller-manager","system:authenticated"]},"sourceIPs":["::1"],"userAgent":"cluster-kube-controller-manager-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"configmaps","namespace":"openshift-kube-controller-manager","name":"cert-recovery-controller-lock","uid":"5c57190b-6993-425d-8101-8337e48c7548","apiVersion":"v1","resourceVersion":"574307"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2020-04-02T08:27:20.200962Z","stageTimestamp":"2020-04-02T08:27:20.206710Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"system:openshift:operator:kube-controller-manager-recovery\" of ClusterRole \"cluster-admin\" to ServiceAccount \"localhost-recovery-client/openshift-kube-controller-manager\""}}
```


10.2. 監査ログの表示

それぞれのコントロールプレーンノードについて OpenShift API サーバー、Kubernetes API サーバー、OpenShift OAuth API サーバー、および OpenShift OAuth サーバーのログを表示できます。

手順

監査ログを表示するには、以下を実行します。

- OpenShift API サーバーの監査ログを表示します。

  a. 各コントロールプレーンノードで利用可能な OpenShift API サーバー監査ログをリスト表示します。

  ```
  $ oc adm node-logs --role=master --path=openshift-apiserver/
  ```

  出力例

  ```
  ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T00-12-19.834.log
  ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
  ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T00-11-49.835.log
  ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
  ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T00-13-00.128.log
  ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
  ```

  b. ノード名とログ名を指定して、特定の OpenShift API サーバー監査ログを表示します。

  ```
  $ oc adm node-logs <node_name> --path=openshift-apiserver/<log_name>
  ```

  以下に例を示します。

  ```
  $ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=openshift-apiserver/audit-2021-03-09T00-12-19.834.log
  ```

  出力例

  ```
  {"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"381acf6d-5f30-4c7d-8175-c9c317ae5893","stage":"ResponseComplete","requestURI":"/metrics","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","uid":"825b60a0-3976-4861-a342-3b2b561e8f82","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"]},"sourceIPs":["10.129.2.6"],"userAgent":"Prometheus/2.23.0","responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T18:02:04.086545Z","stageTimestamp":"2021-03-08T18:02:04.107102Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"prometheus-k8s\" of ClusterRole \"prometheus-k8s\" to ServiceAccount \"prometheus-k8s/openshift-monitoring\""}}
  ```

- Kubernetes API サーバーの監査ログを表示します。

  a. 各コントロールプレーンノードで利用可能な Kubernetes API 監査サーバーログをリスト表示します。

```
$ oc adm node-logs --role=master --path=kube-apiserver/
```

出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T14-07-27.129.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T19-24-22.620.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T18-37-07.511.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. ノード名とログ名を指定して、特定の Kubernetes API 監査サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=kube-apiserver/<log_name>
```

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=kube-apiserver/audit-
2021-03-09T14-07-27.129.log
```

出力例


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"cfce8a0b-b5f5-
4365-8c9f-
79c1227d10f9","stage":"ResponseComplete","requestURI":"/api/v1/namespaces/openshift-
kube-scheduler/serviceaccounts/openshift-kube-scheduler-sa","verb":"get","user":
{"username":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-
scheduler-operator","uid":"2574b041-f3c8-44e6-a057-baef7aa81516","groups":
["system:serviceaccounts","system:serviceaccounts:openshift-kube-scheduler-
operator","system:authenticated"]},"sourceIPs":["10.128.0.8"],"userAgent":"cluster-kube-
scheduler-operator/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":
{"resource":"serviceaccounts","namespace":"openshift-kube-
scheduler","name":"openshift-kube-scheduler-sa","apiVersion":"v1"},"responseStatus":
{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-
08T18:06:42.512619Z","stageTimestamp":"2021-03-
08T18:06:42.516145Z","annotations":{"authentication.k8s.io/legacy-
token":"system:serviceaccount:openshift-kube-scheduler-operator:openshift-kube-
scheduler-
operator","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC:
allowed by ClusterRoleBinding \"system:openshift:operator:cluster-kube-scheduler-
operator\" of ClusterRole \"cluster-admin\" to ServiceAccount \"openshift-kube-scheduler-
operator/openshift-kube-scheduler-operator\""}}
```


- OpenShift OAuth API サーバーの監査ログを表示します。

a. 各コントロールプレーンノードで利用可能な OpenShift OAuth API 監査サーバーログをリスト表示します。

```
$ oc adm node-logs --role=master --path=oauth-apiserver/
```

出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2021-03-09T13-06-26.128.log
```

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2021-03-09T18-23-21.619.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2021-03-09T17-36-06.510.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. ノード名とログ名を指定して、特定の OpenShift OAuth API 監査サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=oauth-apiserver/<log_name>
```

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-apiserver/audit-2021-03-09T13-06-26.128.log
```

出力例


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"dd4c44e2-3ea1-4830-9ab7-c91a5f1388d6","stage":"ResponseComplete","requestURI":"/apis/user.openshift.io/v1/users/~","verb":"get","user":{"username":"system:serviceaccount:openshift-monitoring:prometheus-k8s","groups":["system:serviceaccounts","system:serviceaccounts:openshift-monitoring","system:authenticated"]},"sourceIPs":["10.0.32.4","10.128.0.1"],"userAgent":"dockerregistry/v0.0.0 (linux/amd64) kubernetes/$Format","objectRef":{"resource":"users","name":"~","apiGroup":"user.openshift.io","apiVersion":"v1"},"responseStatus":{"metadata":{},"code":200},"requestReceivedTimestamp":"2021-03-08T17:47:43.653187Z","stageTimestamp":"2021-03-08T17:47:43.660187Z","annotations":{"authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":"RBAC: allowed by ClusterRoleBinding \"basic-users\" of ClusterRole \"basic-user\" to Group \"system:authenticated\""}}
```


- OpenShift OAuth サーバーの監査ログを表示します。

a. 各コントロールプレーンノードで利用可能な OpenShift OAuth サーバーログをリスト表示します。

```
$ oc adm node-logs --role=master --path=oauth-server/
```

出力例

```
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit-2022-05-11T18-57-32.395.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-0 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit-2022-05-11T19-07-07.021.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-1 audit.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit-2022-05-11T19-06-51.844.log
ci-ln-m0wpfjb-f76d1-vnb5x-master-2 audit.log
```

b. ノード名とログ名を指定して、特定の OpenShift OAuth サーバーログを表示します。

```
$ oc adm node-logs <node_name> --path=oauth-server/<log_name>
```

■

以下に例を示します。

```
$ oc adm node-logs ci-ln-m0wpfjb-f76d1-vnb5x-master-0 --path=oauth-server/audit-
2022-05-11T18-57-32.395.log
```

出力例


```
{"kind":"Event","apiVersion":"audit.k8s.io/v1","level":"Metadata","auditID":"13c20345-
f33b-4b7d-b3b6-
e7793f805621","stage":"ResponseComplete","requestURI":"/login","verb":"post","user":
{"username":"system:anonymous","groups":["system:unauthenticated"]},"sourceIPs":
["10.128.2.6"],"userAgent":"Mozilla/5.0 (X11; Linux x86_64; rv:91.0) Gecko/20100101
Firefox/91.0","responseStatus":{"metadata":
{},"code":302},"requestReceivedTimestamp":"2022-05-
11T17:31:16.280155Z","stageTimestamp":"2022-05-
11T17:31:16.297083Z","annotations":
{"authentication.openshift.io/decision":"error","authentication.openshift.io/username":"kubea
dmin","authorization.k8s.io/decision":"allow","authorization.k8s.io/reason":""}}
```


authentication.openshift.io/decision アノテーションに使用できる値は、**allow**、**deny**、または **error** です。

10.3. 監査ログのフィルター

jq または別の JSON 解析ツールを使用して、API サーバー監査ログをフィルターできます。

注記

API サーバー監査ログに記録する情報量は、設定される監査ログポリシーで制御できます。

以下の手順では、**jq** を使用してコントロールプレーンノード **node-1.example.com** で監査ログをフィルターする例を示します。**jq** の使用に関する詳細は、jq Manual を参照してください。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。
- **jq** がインストールされている。

手順

- OpenShift API サーバー監査ログをユーザーでフィルターします。

```
$ oc adm node-logs node-1.example.com  \
  --path=openshift-apiserver/audit.log \
  | jq 'select(.user.username == "myusername")'
```

- OpenShift API サーバー監査ログをユーザーエージェントでフィルターします。

```
$ oc adm node-logs node-1.example.com  \
```

```
  --path=openshift-apiserver/audit.log \
  | jq 'select(.userAgent == "cluster-version-operator/v0.0.0 (linux/amd64)
kubernetes/$Format")'
```

- Kubernetes API サーバー監査ログを特定の API バージョンでフィルターし、ユーザーエージェントのみを出力します。

```
$ oc adm node-logs node-1.example.com  \
  --path=kube-apiserver/audit.log \
  | jq 'select(.requestURI | startswith("/apis/apiextensions.k8s.io/v1beta1")) | .userAgent'
```

- 動詞を除外して OpenShift OAuth API サーバー監査ログをフィルターします。

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-apiserver/audit.log \
  | jq 'select(.verb != "get")'
```

- ユーザー名を識別し、エラーで失敗したイベントで OpenShift OAuth サーバー監査ログをフィルタリングします。

```
$ oc adm node-logs node-1.example.com  \
  --path=oauth-server/audit.log \
  | jq 'select(.annotations["authentication.openshift.io/username"] != null and
.annotations["authentication.openshift.io/decision"] == "error")'
```

10.4. 監査ログの収集

must-gather ツールを使用して、クラスターをデバッグするための監査ログを収集できます。このログは、確認したり、Red Hat サポートに送信したりできます。

手順

1. **-- /usr/bin/gather_audit_logs** を指定して **oc adm must-gather** コマンドを実行します。

```
$ oc adm must-gather -- /usr/bin/gather_audit_logs
```

2. 作業ディレクトリーに作成された **must-gather** ディレクトリーから圧縮ファイルを作成します。たとえば、Linux オペレーティングシステムを使用するコンピューターで以下のコマンドを実行します。

```
$ tar cvaf must-gather.tar.gz must-gather.local.472290403699006248 (1)
```

(1) **must-gather-local.472290403699006248** は、実際のディレクトリー名に置き換えます。

3. Red Hat カスタマーポータルの **カスタマーサポート ページ** で、圧縮ファイルをサポートケースに添付します。

10.5. 関連情報

- must-gather ツール

- API 監査ログイベントの構造
- 監査ログポリシーの設定
- ログ転送

第11章 監査ログポリシーの設定

使用する監査ログポリシープロファイルを選択して、API サーバー監査ログに記録する情報量を制御できます。

11.1. 監査ログポリシープロファイルについて

監査ログプロファイルは、OpenShift API サーバー、Kubernetes API サーバー、OpenShift OAuth API サーバー、および OpenShift OAuth サーバーに送信される要求をログに記録する方法を定義します。

OpenShift Container Platform は、以下の事前定義された監査ポリシープロファイルを提供します。

プロファイル	説明
Default	読み取りおよび書き込み要求のメタデータのみをログに記録します。OAuth アクセストークン要求を除く要求の本文はログに記録されません。これはデフォルトポリシーになります。
WriteRequestBodies	すべてのリクエストのメタデータのログに加えて、API サーバーへのすべての書き込みリクエスト (**create**、**update**、**patch**、**delete**、**deletecollection**) のリクエスト本文をログに記録します。このプロファイルには、**Default** プロファイルよりも多くのリソースのオーバーヘッドがあります。[1]
AllRequestBodies	すべての要求のメタデータをロギングする以外にも、API サーバーへの読み取りおよび書き込み要求ごとに要求の本文をログに記録します (**get**、**list**、**create**、**update**、**patch**)。このプロファイルには最も多くのリソースのオーバーヘッドがあります。[1]
None	OAuth アクセストークン要求や OAuth 承認トークン要求などの要求はログに記録されません。このプロファイルが設定されている場合、カスタムルールは無視されます。 **警告** 問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にしないようにしてください。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現することを推奨します。

1. **Secret**、**Route**、**OAuthClient** オブジェクトなどの機密リソースは、メタデータレベルでのみログ記録されます。OpenShift OAuth サーバーイベントは、メタデータレベルでのみログに記録されます。

デフォルトで、OpenShift Container Platform は **Default** 監査ログプロファイルを使用します。リクエスト本文もログに記録する別の監査ポリシープロファイルを使用することもできますが、CPU、メモリー、I/O などのリソース使用量が増加することに注意してください。

11.2. 監査ログポリシーの設定

API サーバーに送信される要求をログに記録する際に使用する監査ログポリシーを設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.profile** フィールドを更新します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: WriteRequestBodies 1
```

1 **Default**、**WriteRequestBodies**、**AllRequestBodies** または **None** に設定されます。デフォルトのプロファイルは **Default** です。

警告

問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現することを推奨します。

3. 変更を適用するためにファイルを保存します。

検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) この例では、最新のリビジョン番号は **12** です。

出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

11.3. カスタムルールによる監査ログポリシーの設定

カスタムルールを定義する監査ログポリシーを設定できます。複数のグループを指定し、対象のグループに使用するプロファイルを定義できます。

これらのカスタムルールは最上位のプロファイルフィールドよりも優先されます。カスタムルールはトップダウンで評価され、最初に一致するものが適用されます。

重要

最上位のプロファイルフィールドを **None** に設定すると、Kubernetes API サーバーなどの API サーバーがカスタムルールを無視し、監査ロギングを無効にします。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.customRules** フィールドを追加します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    customRules: (1)
    - group: system:authenticated:oauth
```

```
    profile: WriteRequestBodies
  - group: system:authenticated
    profile: AllRequestBodies
  profile: Default 2
```

1 1 つ以上のグループを追加し、そのグループに使用するプロファイルを指定します。これらのカスタムルールは最上位のプロファイルフィールドよりも優先されます。カスタムルールはトップダウンで評価され、最初に一致するものが適用されます。

2 **Default**、**WriteRequestBodies**、または **AllRequestBodies** に設定されます。この最上位のプロファイルフィールドを設定しない場合、デフォルトは **Default** プロファイルになります。

3. 変更を適用するためにファイルを保存します。

検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

  Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 1
```

  1 この例では、最新のリビジョン番号は **12** です。

  出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

  - **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
  - **2 nodes are at revision 11; 1 nodes are at revision 12**

11.4. 監査ロギングの無効化

OpenShift Container Platform の監査ロギングを無効にできます。監査ロギングを無効にする場合には、OAuth アクセストークン要求および OAuth 認証トークン要求もログに記録されません。

> **警告**
>
> 問題のトラブルシューティング時に有用なデータが記録されないリスクを完全に理解していない限り、**None** プロファイルを使用して監査ロギングを無効にすることは推奨していません。監査ロギングを無効にしてサポートが必要な状況が生じた場合は、適切にトラブルシューティングを行うために監査ロギングを有効にし、問題を再現することを推奨します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver cluster
```

2. **spec.audit.profile** フィールドを **None** に設定します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
...
spec:
  audit:
    profile: None
```

> **注記**
>
> また、**spec.audit.customRules** フィールドにカスタムルールを指定して、特定のグループに関する監査ロギングのみを無効にすることもできます。

3. 変更を適用するためにファイルを保存します。

検証

- Kubernetes API サーバー Pod の新規リビジョンがロールアウトされていることを確認します。すべてのノードが新規リビジョンに更新されるまで数分かかる場合があります。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="NodeInstallerProgressing")]}{.reason}{"\n"}{.message}{"\n"}'
```

Kubernetes API サーバーの **NodeInstallerProgressing** 状況条件を確認し、すべてのノードが最新のリビジョンであることを確認します。更新が正常に実行されると、この出力には **AllNodesAtLatestRevision** が表示されます。

```
AllNodesAtLatestRevision
3 nodes are at revision 12 (1)
```

(1) この例では、最新のリビジョン番号は **12** です。

出力に以下のようなメッセージが表示される場合は、更新が進行中です。数分待機した後に再試行します。

- **3 nodes are at revision 11; 0 nodes have achieved new revision 12**
- **2 nodes are at revision 11; 1 nodes are at revision 12**

第12章 TLS セキュリティープロファイルの設定

TLS セキュリティープロファイルは、サーバーへの接続時に、クライアントが使用できる暗号を規制する方法をサーバーに提供します。これにより、OpenShift Container Platform コンポーネントは暗号化ライブラリーを使用するようになり、既知の安全ではないプロトコル、暗号、またはアルゴリズムを拒否します。

クラスター管理者は、以下のコンポーネントごとに使用する TLS セキュリティープロファイルを選択できます。

- Ingress コントローラー
- コントロールプレーン
  これには、Kubernetes API サーバー、Kubernetes コントローラーマネージャー、Kubernetes スケジューラー、OpenShift API サーバー、OpenShift OAuth API サーバー、OpenShift OAuth サーバーおよび etcd が含まれます。
- kubelet (Kubernetes API サーバーの HTTP サーバーとして機能する場合)

12.1. TLS セキュリティープロファイルについて

TLS (Transport Layer Security) セキュリティープロファイルを使用して、さまざまな OpenShift Container Platform コンポーネントに必要な TLS 暗号を定義できます。OpenShift Container Platform の TLS セキュリティープロファイルは、Mozilla が推奨する設定 に基づいています。

コンポーネントごとに、以下の TLS セキュリティープロファイルのいずれかを指定できます。

表12.1 TLS セキュリティープロファイル

プロファイル	説明
Old	このプロファイルは、レガシークライアントまたはライブラリーでの使用を目的としています。このプロファイルは、Old 後方互換性 の推奨設定に基づいています。 **Old** プロファイルには、最小 TLS バージョン 1.0 が必要です。 **注記** Ingress Controller の場合、TLS の最小バージョンは 1.0 から 1.1 に変換されます。

プロファイル	説明
Intermediate	このプロファイルは、Ingress Controller、kubelet、およびコントロールプレーンのデフォルトの TLS セキュリティープロファイルです。このプロファイルは、Intermediate 互換性 の推奨設定に基づいています。 **Intermediate** プロファイルには、最小 TLS バージョン 1.2 が必要です。 **注記** このプロファイルは、大多数のクライアントに推奨される設定です。
Modern	このプロファイルは、下位互換性を必要としない最新のクライアントで使用することを目的としています。このプロファイルは、Modern compatibility の推奨設定に基づいています。 **Modern** プロファイルには、最小 TLS バージョン 1.3 が必要です。
Custom	このプロファイルを使用すると、使用する TLS バージョンと暗号を定義できます。 **警告** 無効な設定により問題が発生する可能性があるため、**Custom** プロファイルを使用する際には注意してください。

注記

事前定義されたプロファイルタイプのいずれかを使用する場合、有効なプロファイル設定はリリース間で変更される可能性があります。たとえば、リリース X.Y.Z にデプロイされた Intermediate プロファイルを使用する仕様がある場合、リリース X.Y.Z+1 へのアップグレードにより、新規のプロファイル設定が適用され、ロールアウトが生じる可能性があります。

12.2. TLS セキュリティープロファイルの詳細表示

Ingress コントローラー、コントロールプレーン、および kubelet のコンポーネントごとに事前定義された TLS セキュリティープロファイルの最小 TLS バージョンおよび暗号を表示できます。

重要

プロファイルの最小 TLS バージョンと暗号のリストは、コンポーネントによって異なる場合があります。

手順

- 特定の TLS セキュリティープロファイルの詳細を表示します。

```
$ oc explain <component>.spec.tlsSecurityProfile.<profile> 1
```

1 **<component>** には、**ingresscontroller**、**apiserver** または **kubeletconfig** を指定します。**<profile>** には、**old**、**intermediate** または **custom** を指定します。

たとえば、コントロールプレーンの **intermediate** プロファイルに含まれる暗号を確認するには、以下を実行します。

```
$ oc explain apiserver.spec.tlsSecurityProfile.intermediate
```

出力例

```
KIND:     APIServer
VERSION:  config.openshift.io/v1

DESCRIPTION:
    intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended
.29
    and looks like this (yaml):
    ciphers: - TLS_AES_128_GCM_SHA256 - TLS_AES_256_GCM_SHA384 -
    TLS_CHACHA20_POLY1305_SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256 -
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES256-GCM-SHA384 -
    ECDHE-RSA-AES256-GCM-SHA384 - ECDHE-ECDSA-CHACHA20-POLY1305 -
    ECDHE-RSA-CHACHA20-POLY1305 - DHE-RSA-AES128-GCM-SHA256 -
    DHE-RSA-AES256-GCM-SHA384 minTLSVersion: TLSv1.2
```

- コンポーネントの **tlsSecurityProfile** フィールドの詳細をすべて表示します。

```
$ oc explain <component>.spec.tlsSecurityProfile 1
```

1 **<component>** には、**ingresscontroller**、**apiserver** または **kubeletconfig** を指定します。

たとえば、Ingress コントローラーの **tlsSecurityProfile** フィールドの詳細をすべて確認するには、以下を実行します。

```
$ oc explain ingresscontroller.spec.tlsSecurityProfile
```

出力例

```
KIND:     IngressController
VERSION:  operator.openshift.io/v1

RESOURCE: tlsSecurityProfile <Object>

DESCRIPTION:
```

```
    ...

FIELDS:
  custom <>
    custom is a user-defined TLS security profile. Be extremely careful using a
    custom profile as invalid configurations can be catastrophic. An example
    custom profile looks like this:
    ciphers: - ECDHE-ECDSA-CHACHA20-POLY1305 - ECDHE-RSA-CHACHA20-
POLY1305 -
    ECDHE-RSA-AES128-GCM-SHA256 - ECDHE-ECDSA-AES128-GCM-SHA256
minTLSVersion:
    TLSv1.1

  intermediate <>
    intermediate is a TLS security profile based on:

https://wiki.mozilla.org/Security/Server_Side_TLS#Intermediate_compatibility_.28recommended
.29
    and looks like this (yaml):
    ... (1)

  modern <>
    modern is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Modern_compatibility and
    looks like this (yaml):
    ... (2)
    NOTE: Currently unsupported.

  old <>
    old is a TLS security profile based on:
    https://wiki.mozilla.org/Security/Server_Side_TLS#Old_backward_compatibility
    and looks like this (yaml):
    ... (3)

  type <string>
    ...
```

1. ここでは、**intermediate** プロファイルの暗号および最小バージョンをリスト表示します。
2. **modern** プロファイルの暗号と最小バージョンをリスト表示します。
3. ここでは、**old** プロファイルの暗号および最小バージョンをリスト表示します。

12.3. INGRESS CONTROLLER の TLS セキュリティープロファイルの設定

Ingress Controller の TLS セキュリティープロファイルを設定するには、**IngressController** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合、デフォルト値は API サーバーに設定された TLS セキュリティープロファイルに基づいています。

Old TLS のセキュリティープロファイルを設定するサンプル IngressController CR

```
apiVersion: operator.openshift.io/v1
kind: IngressController
```

```
...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
...
```

TLS セキュリティープロファイルは、Ingress Controller の TLS 接続の最小 TLS バージョンと TLS 暗号を定義します。

設定された TLS セキュリティープロファイルの暗号と最小 TLS バージョンは、**Status.Tls Profile** 配下の **IngressController** カスタムリソース (CR) と **Spec.Tls Security Profile** 配下の設定された TLS セキュリティープロファイルで確認できます。**Custom** TLS セキュリティープロファイルの場合、特定の暗号と最小 TLS バージョンは両方のパラメーターの下に一覧表示されます。

注記

HAProxy Ingress Controller イメージは、TLS **1.3** と **Modern** プロファイルをサポートしています。

また、Ingress Operator は TLS **1.0** の **Old** または **Custom** プロファイルを **1.1** に変換します。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **openshift-ingress-operator** プロジェクトの **IngressController** CR を編集して、TLS セキュリティープロファイルを設定します。

   ```
   $ oc edit IngressController default -n openshift-ingress-operator
   ```

2. **spec.tlsSecurityProfile** フィールドを追加します。

   Custom プロファイルのサンプル IngressController CR

   ```
   apiVersion: operator.openshift.io/v1
   kind: IngressController
    ...
   spec:
     tlsSecurityProfile:
       type: Custom 1
       custom: 2
         ciphers: 3
         - ECDHE-ECDSA-CHACHA20-POLY1305
         - ECDHE-RSA-CHACHA20-POLY1305
         - ECDHE-RSA-AES128-GCM-SHA256
         - ECDHE-ECDSA-AES128-GCM-SHA256
         minTLSVersion: VersionTLS11
    ...
   ```

1. TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。

2. 選択したタイプに適切なフィールドを指定します。
   - **old: {}**
   - **intermediate: {}**
   - **custom:**

3. **custom** タイプには、TLS 暗号のリストと最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

検証

- **IngressController** CR にプロファイルが設定されていることを確認します。

```
$ oc describe IngressController default -n openshift-ingress-operator
```

出力例

```
Name:         default
Namespace:    openshift-ingress-operator
Labels:       <none>
Annotations:  <none>
API Version:  operator.openshift.io/v1
Kind:         IngressController
 ...
Spec:
 ...
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:               Custom
 ...
```

12.4. コントロールプレーンの TLS セキュリティープロファイルの設定

コントロールプレーンの TLS セキュリティープロファイルを設定するには、**APIServer** カスタムリソース (CR) を編集して、事前定義済みまたはカスタムの TLS セキュリティープロファイルを指定します。**APIServer** CR に TLS セキュリティープロファイルを設定すると、設定は以下のコントロールプレーンのコンポーネントに伝播されます。

- Kubernetes API サーバー
- Kubernetes コントローラーマネージャー

- Kubernetes スケジューラー
- OpenShift API サーバー
- OpenShift OAuth API サーバー
- OpenShift OAuth サーバー
- etcd

TLS セキュリティープロファイルが設定されていない場合には、TLS セキュリティープロファイルは **Intermediate** になります。

> **注記**
>
> Ingress コントローラーのデフォルトの TLS セキュリティープロファイルは API サーバーの TLS セキュリティープロファイルに基づいています。

Old TLS のセキュリティープロファイルを設定するサンプル APIServer CR

```
apiVersion: config.openshift.io/v1
kind: APIServer
 ...
spec:
  tlsSecurityProfile:
    old: {}
    type: Old
 ...
```

TLS セキュリティープロファイルは、コントロールプレーンのコンポーネントとの通信に必要な TLS の最小バージョンと TLS 暗号を定義します。

設定した TLS セキュリティープロファイルは、**Spec.Tls Security Profile** の **APIServer** カスタムリソース (CR) で確認できます。**Custom** TLS セキュリティープロファイルの場合には、特定の暗号と最小 TLS バージョンが一覧表示されます。

> **注記**
>
> コントロールプレーンは、最小 TLS バージョンとして TLS **1.3** をサポートしません。**Modern** プロファイルは TLS **1.3** を必要とするため、サポート対象外です。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. デフォルトの **APIServer** CR を編集して TLS セキュリティープロファイルを設定します。

```
$ oc edit APIServer cluster
```

2. **spec.tlsSecurityProfile** フィールドを追加します。

Custom プロファイルの APIServer CR のサンプル

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  name: cluster
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
      ciphers: 3
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
```

1 TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。

2 選択したタイプに適切なフィールドを指定します。

- **old: {}**
- **intermediate: {}**
- **custom:**

3 **custom** タイプには、TLS 暗号のリストと最小許容 TLS バージョンを指定します。

3. 変更を適用するためにファイルを保存します。

検証

- TLS セキュリティープロファイルが **APIServer** CR に設定されていることを確認します。

```
$ oc describe apiserver cluster
```

出力例

```
Name:         cluster
Namespace:
 ...
API Version:  config.openshift.io/v1
Kind:         APIServer
 ...
Spec:
  Audit:
    Profile:  Default
  Tls Security Profile:
    Custom:
      Ciphers:
        ECDHE-ECDSA-CHACHA20-POLY1305
```

```
        ECDHE-RSA-CHACHA20-POLY1305
        ECDHE-RSA-AES128-GCM-SHA256
        ECDHE-ECDSA-AES128-GCM-SHA256
      Min TLS Version:  VersionTLS11
    Type:             Custom
...
```

- TLS セキュリティープロファイルが **etcd** CR に設定されていることを確認します。

```
$ oc describe etcd cluster
```

出力例

```
Name:        cluster
Namespace:
...
API Version:  operator.openshift.io/v1
Kind:        Etcd
...
Spec:
  Log Level:       Normal
  Management State:  Managed
  Observed Config:
    Serving Info:
      Cipher Suites:
        TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256
        TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256
        TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384
        TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384
        TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256
        TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256
      Min TLS Version:          VersionTLS12
...
```

12.5. KUBELET の TLS セキュリティープロファイルの設定

HTTP サーバーとしての動作時に kubelet の TLS セキュリティープロファイルを設定するには、**KubeletConfig** カスタムリソース (CR) を作成して特定のノード用に事前定義済みの TLS セキュリティープロファイルまたはカスタム TLS セキュリティープロファイルを指定します。TLS セキュリティープロファイルが設定されていない場合には、TLS セキュリティープロファイルは **Intermediate** になります。

kubelet はその HTTP/GRPC サーバーを使用して Kubernetes API サーバーと通信し、コマンドを Pod に送信して kubelet 経由で Pod で exec コマンドを実行します。

ワーカーノードで Old TLS セキュリティープロファイルを設定する KubeletConfig CR のサンプル

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
# ...
spec:
  tlsSecurityProfile:
    old: {}
```

```
    type: Old
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: ""
# ...
```

設定済みのノードの **kubelet.conf** ファイルで、設定済みの TLS セキュリティープロファイルの暗号化および最小 TLS セキュリティープロファイルを確認できます。

前提条件

- **cluster-admin** ロールを持つユーザーとして OpenShift Container Platform にログインしている。

手順

1. **KubeletConfig** CR を作成し、TLS セキュリティープロファイルを設定します。

Custom プロファイル用の KubeletConfig CR のサンプル

```
apiVersion: machineconfiguration.openshift.io/v1
kind: KubeletConfig
metadata:
  name: set-kubelet-tls-security-profile
spec:
  tlsSecurityProfile:
    type: Custom 1
    custom: 2
      ciphers: 3
      - ECDHE-ECDSA-CHACHA20-POLY1305
      - ECDHE-RSA-CHACHA20-POLY1305
      - ECDHE-RSA-AES128-GCM-SHA256
      - ECDHE-ECDSA-AES128-GCM-SHA256
      minTLSVersion: VersionTLS11
  machineConfigPoolSelector:
    matchLabels:
      pools.operator.machineconfiguration.openshift.io/worker: "" 4
#...
```

1 TLS セキュリティープロファイルタイプ (**Old**、**Intermediate**、または **Custom**) を指定します。デフォルトは **Intermediate** です。

2 選択したタイプに適切なフィールドを指定します。

- **old: {}**
- **intermediate: {}**
- **custom:**

3 **custom** タイプには、TLS 暗号のリストと最小許容 TLS バージョンを指定します。

4 オプション: TLS セキュリティープロファイルを適用するノードのマシン設定プールラベルを指定します。

2. **KubeletConfig** オブジェクトを作成します。

```
$ oc create -f <filename>
```

クラスター内のワーカーノードの数によっては、設定済みのノードが 1 つずつ再起動されるのを待機します。

検証

プロファイルが設定されていることを確認するには、ノードが **Ready** になってから以下の手順を実行します。

1. 設定済みノードのデバッグセッションを開始します。

```
$ oc debug node/<node_name>
```

2. **/host** をデバッグシェル内のルートディレクトリーとして設定します。

```
sh-4.4# chroot /host
```

3. **kubelet.conf** ファイルを表示します。

```
sh-4.4# cat /etc/kubernetes/kubelet.conf
```

出力例

```
  "kind": "KubeletConfiguration",
  "apiVersion": "kubelet.config.k8s.io/v1beta1",
#...
  "tlsCipherSuites": [
    "TLS_ECDHE_ECDSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_RSA_WITH_AES_128_GCM_SHA256",
    "TLS_ECDHE_ECDSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_RSA_WITH_AES_256_GCM_SHA384",
    "TLS_ECDHE_ECDSA_WITH_CHACHA20_POLY1305_SHA256",
    "TLS_ECDHE_RSA_WITH_CHACHA20_POLY1305_SHA256"
  ],
  "tlsMinVersion": "VersionTLS12",
#...
```

第13章 SECCOMP プロファイルの設定

OpenShift Container Platform コンテナーまたは Pod は、1 つ以上の明確に定義されたタスクを実行するアプリケーションを 1 つ実行します。アプリケーションには通常、基礎となるオペレーティングシステムカーネル API の小規模なサブセットのみが必要です。セキュアコンピューティングモードである seccomp は Linux カーネル機能で、これを使用して、コンテナーで実行されているプロセスを制限して、利用可能なシステム呼び出しのサブセットだけが使用できるようにできます。

restricted-v2 SCC は、4.16 で新しく作成されたすべての Pod に適用されます。これらの Pod には、デフォルトの seccomp プロファイル **runtime/default** が適用されます。

seccomp プロファイルは、ディスクに JSON ファイルとして保存されます。

重要

seccomp プロファイルは特権付きコンテナーに適用できません。

13.1. POD に適用されるデフォルトの SECCOMP プロファイルの確認

OpenShift Container Platform には、デフォルトの seccomp プロファイルが同梱されており、**runtime/default** として参照されます。4.16 では、新しく作成された Pod の Security Context Constraint (SCC) が **restricted-v2** に設定され、デフォルトの seccomp プロファイルが Pod に適用されます。

手順

1. 次のコマンドを実行して、Pod に設定されている Security Context Constraint (SCC) とデフォルトの seccomp プロファイルを確認できます。

   a. namespace で実行されている Pod を確認します。

   ```
   $ oc get pods -n <namespace>
   ```

   たとえば、**workshop** namespace で実行されている Pod を確認するには、次を実行します。

   ```
   $ oc get pods -n workshop
   ```

   出力例

   ```
   NAME                READY   STATUS      RESTARTS   AGE
   parksmap-1-4xkwf    1/1     Running     0          2m17s
   parksmap-1-deploy   0/1     Completed   0          2m22s
   ```

   b. Pod を調べます。

   ```
   $ oc get pod parksmap-1-4xkwf -n workshop -o yaml
   ```

   出力例

   ```
   apiVersion: v1
   kind: Pod
   ```

```
metadata:
  annotations:
    k8s.v1.cni.cncf.io/network-status: |-
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.131.0.18"
          ],
          "default": true,
          "dns": {}
      }]
    k8s.v1.cni.cncf.io/network-status: |-
      [{
          "name": "openshift-sdn",
          "interface": "eth0",
          "ips": [
              "10.131.0.18"
          ],
          "default": true,
          "dns": {}
      }]
    openshift.io/deployment-config.latest-version: "1"
    openshift.io/deployment-config.name: parksmap
    openshift.io/deployment.name: parksmap-1
    openshift.io/generated-by: OpenShiftWebConsole
    openshift.io/scc: restricted-v2 (1)
    seccomp.security.alpha.kubernetes.io/pod: runtime/default (2)
```

1. ワークロードが別の SCC にアクセスできない場合、**restricted-v2** SCC がデフォルトで追加されます。

2. 4.16 で新しく作成された Pod には、SCC で義務付けられたとおり、**runtime/default** に設定された seccomp プロファイルがあります。

13.1.1. アップグレードされたクラスター

4.16 にアップグレードされたクラスターでは、すべての認証済みユーザーが **restricted** および **restricted-v2** SCC にアクセスできます。

たとえば、アップグレード時に OpenShift Container Platform v4.10 クラスターで **restricted** となった SCC によって許可されたワークロードは、**restricted-v2** によって許可される場合があります。これは、**restricted-v2** が **restricted** と **restricted-v2** の間のより制限的な SCC であるためです。

注記

ワークロードは **retricted-v2** で実行できる必要があります。

逆に、**privilegeEscalation: true** を必要とするワークロードでは、このワークロードは、認証されたユーザーが使用できる **restricted** SCC を引き続き持ちます。これは、**restricted-v2** が **privilegeEscalation** を許可しないためです。

13.1.2. 新しくインストールされたクラスター

新しくインストールされた OpenShift Container Platform v4.11 以降のクラスターの場合、**restricted-v2** は、認証されたユーザーが使用できる SCC として **restricted** SCC を置き換えます。デフォルトで認証されたユーザーが使用できる SCC は **restricted-v2** のみであるため、**privilegeEscalation: true** を持つワークロードはクラスターへの参加が許可されません。

機能の **privilegeEscalation** は、**restricted** では許可されていますが、**restricted-v2** では許可されていません。**restricted** SCC で許可されていたよりも多くの機能が、**restricted-v2** によって拒否されます。

新しくインストールされた OpenShift Container Platform v4.11 以降のクラスターには、**privilegeEscalation: true** を持つワークロードが許可される場合があります。RoleBinding を使用して、ワークロードを実行している ServiceAccount (またはこのワークロードを許可できるその他の SCC) に **restricted** SCC へのアクセスを許可するには、次のコマンドを実行します。

```
$ oc -n <workload-namespace> adm policy add-scc-to-user <scc-name> -z <serviceaccount_name>
```

OpenShift Container Platform 4.16 では、Pod アノテーション **seccomp.security.alpha.kubernetes.io/pod: runtime/default** および **container.seccomp.security.alpha.kubernetes.io/<container_name>: runtime/default** を追加する機能が非推奨になりました。

13.2. カスタム SECCOMP プロファイルの設定

カスタム seccomp プロファイルを設定すると、アプリケーション要件に基づいてフィルターを更新できます。これにより、クラスター管理者は OpenShift Container Platform で実行されるワークロードのセキュリティーをより詳細に制御できます。

Seccomp セキュリティープロファイルには、プロセスが実行できるシステムコール (syscall) がリストされています。権限は、**write** などの操作をシステム全体で制限する SELinux よりも広範です。

13.2.1. seccomp プロファイルの作成

MachineConfig オブジェクトを使用してプロファイルを作成できます。

Seccomp は、コンテナー内のシステムコール (syscall) を制限して、アプリケーションのアクセスを制限できます。

前提条件

- クラスター管理者パーミッションがある。
- カスタム Security Context Constraints (SCC) を作成している。詳細は、**関連情報** を参照してください。

手順

- **MachineConfig** オブジェクトを作成します。

```
apiVersion: machineconfiguration.openshift.io/v1
kind: MachineConfig
metadata:
  labels:
```

```
    machineconfiguration.openshift.io/role: worker
  name: custom-seccomp
spec:
  config:
    ignition:
      version: 3.2.0
    storage:
      files:
      - contents:
          source: data:text/plain;charset=utf-8;base64,<hash>
        filesystem: root
        mode: 0644
        path: /var/lib/kubelet/seccomp/seccomp-nostat.json
```

13.2.2. カスタム seccomp プロファイルのセットアップ

前提条件

- クラスター管理者パーミッションがある。
- カスタム Security Context Constraints (SCC) を作成している。詳細は、「関連情報」を参照してください。
- カスタム seccomp プロファイルを作成している。

手順

1. Machine Config を使用してカスタム seccomp プロファイルを **/var/lib/kubelet/seccomp/<custom-name>.json** にアップロードします。詳細な手順は、「関連情報」を参照してください。
2. 作成されたカスタム seccomp プロファイルへの参照を指定してカスタム SCC を更新します。

```
seccompProfiles:
- localhost/<custom-name>.json (1)
```

(1) カスタム seccomp プロファイルの名前を入力します。

13.2.3. カスタム seccomp プロファイルのワークロードへの適用

前提条件

- クラスター管理者はカスタム seccomp プロファイルを設定している。詳細は、「カスタム seccomp プロファイルの設定」を参照してください。

手順

- **securityContext.seccompProfile.type** フィールドを次のように設定して、seccomp プロファイルをワークロードに適用します。

  例

```
spec:
  securityContext:
    seccompProfile:
      type: Localhost
      localhostProfile: <custom-name>.json 1
```

1. カスタム seccomp プロファイルの名前を入力します。

または、Pod アノテーション **seccomp.security.alpha.kubernetes.io/pod: localhost/<custom-name>.json** を使用できます。ただし、この手法は OpenShift Container Platform 4.16 では非推奨になっています。

デプロイメント時に、受付コントローラーは以下を検証します。

- 現在の SCC に対するアノテーションがユーザーロールで許可されている。
- seccomp プロファイルを含む SCC が Pod で許可されている。

SCC が Pod で許可される場合には、kubelet は指定された seccomp プロファイルで Pod を実行します。

重要

seccomp プロファイルがすべてのワーカーノードにデプロイされていることを確認します。

注記

カスタム SCC は、Pod に適切な優先順位で自動的に割り当てられるか、CAP_NET_ADMIN を許可するなど、Pod で必要な他の条件を満たす必要があります。

13.3. 関連情報

- Security Context Constraints の管理
- マシン設定の概要

第14章 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可

14.1. 追加ホストから API サーバーへの JAVASCRIPT ベースのアクセスの許可

デフォルトの OpenShift Container Platform 設定は、Web コンソールが要求を API サーバーに送信することのみを許可します。

別の名前を使用して JavaScript アプリケーションから API サーバーまたは OAuth サーバーにアクセスする必要がある場合、許可する追加のホスト名を設定できます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** リソースを編集します。

```
$ oc edit apiserver.config.openshift.io cluster
```

2. **additionalCORSAllowedOrigins** フィールドを **spec** セクションの下に追加し、1 つ以上の追加のホスト名を指定します。

```
apiVersion: config.openshift.io/v1
kind: APIServer
metadata:
  annotations:
    release.openshift.io/create-only: "true"
  creationTimestamp: "2019-07-11T17:35:37Z"
  generation: 1
  name: cluster
  resourceVersion: "907"
  selfLink: /apis/config.openshift.io/v1/apiservers/cluster
  uid: 4b45a8dd-a402-11e9-91ec-0219944e0696
spec:
  additionalCORSAllowedOrigins:
  - (?i)//my\.subdomain\.domain\.com(:|\z) 1
```

1 ホスト名は Golang 正規表現 として指定されます。これは、API サーバーおよび OAuth サーバーに対する HTTP 要求の CORS ヘッダーに対するマッチングを行うために使用されます。

注記

この例では、以下の構文を使用します。

- **(?i)** は大文字/小文字を区別します。
- // はドメインの開始にピニングし、**http:** または **https:** の後のダブルスラッシュに一致します。
- **\.** はドメイン名のドットをエスケープします。
- **(:|\z)** はドメイン名 **(\z)** またはポートセパレーター **(:)** の終了部に一致します。

3. 変更を適用するためにファイルを保存します。

第15章 ETCD データの暗号化

15.1. ETCD 暗号化について

デフォルトで、etcd データは OpenShift Container Platform で暗号化されません。クラスターの etcd 暗号化を有効にして、データセキュリティーのレイヤーを追加で提供することができます。たとえば、etcd バックアップが正しくない公開先に公開される場合に機密データが失われないように保護することができます。

etcd の暗号化を有効にすると、以下の OpenShift API サーバーおよび Kubernetes API サーバーリソースが暗号化されます。

- シークレット
- config map
- ルート
- OAuth アクセストークン
- OAuth 認証トークン

etcd 暗号を有効にすると、暗号化キーが作成されます。etcd バックアップから復元するには、これらのキーが必要です。

注記

etcd 暗号化は、キーではなく、値のみを暗号化します。リソースの種類、namespace、およびオブジェクト名は暗号化されません。

バックアップ中に etcd 暗号化が有効になっている場合は、**static_kuberesources_<datetimestamp>.tar.gz** ファイルに etcd スナップショットの暗号化キーが含まれています。セキュリティー上の理由から、このファイルは etcd スナップショットとは別に保存してください。ただし、このファイルは、それぞれの etcd スナップショットから etcd の以前の状態を復元するために必要です。

15.2. サポートされている暗号化の種類

以下の暗号化タイプは、OpenShift Container Platform で etcd データを暗号化するためにサポートされています。

AES-CBC

暗号化を実行するために、PKCS#7 パディングと 32 バイトの鍵を含む AES-CBC を使用します。暗号化キーは毎週ローテーションされます。

AES-GCM

AES-GCM とランダムナンスおよび 32 バイトキーを使用して暗号化を実行します。暗号化キーは毎週ローテーションされます。

15.3. ETCD 暗号化の有効化

etcd 暗号化を有効にして、クラスターで機密性の高いリソースを暗号化できます。

> **警告**
>
> 初期暗号化プロセスが完了するまで、etcd リソースをバックアップしないでください。暗号化プロセスが完了しない場合、バックアップは一部のみ暗号化される可能性があります。
>
> etcd 暗号化を有効にすると、いくつかの変更が発生する可能性があります。
>
> - etcd 暗号化は、いくつかのリソースのメモリー消費に影響を与える可能性があります。
> - リーダーがバックアップを提供する必要があるため、バックアップのパフォーマンスに一時的な影響が生じる場合があります。
> - ディスク I/O は、バックアップ状態を受け取るノードに影響を与える可能性があります。

etcd データベースは、AES-GCM または AES-CBC 暗号化で暗号化できます。

> **注記**
>
> etcd データベースをある暗号化タイプから別の暗号化タイプに移行するには、API サーバーの **spec.encryption.type** フィールドを変更します。etcd データの新しい暗号化タイプへの移行は自動的に行われます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **spec.encryption.type** フィールドを **aesgcm** または **aescbc** に設定します。

```
spec:
  encryption:
    type: aesgcm 1
```

1 AES-CBC 暗号化の場合は **aescbc** に、AES-GCM 暗号化の場合は **aesgcm** に設定します。

3. 変更を適用するためにファイルを保存します。
暗号化プロセスが開始されます。etcd データベースのサイズによっては、このプロセスが完了するまでに 20 分以上かかる場合があります。

4. etcd 暗号化が正常に行われたことを確認します。

a. OpenShift API サーバーの **Encrypted** ステータスを確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: routes.route.openshift.io
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: secrets, configmaps
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

c. OpenShift OAuth API サーバーの **Encrypted** ステータスを確認し、そのリソースが正常に暗号化されたことを確認します。

```
$ oc get authentication.operator.openshift.io -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
```

この出力には、暗号化が正常に実行されると **EncryptionCompleted** が表示されます。

```
EncryptionCompleted
All resources encrypted: oauthaccesstokens.oauth.openshift.io, oauthauthorizetokens.oauth.openshift.io
```

出力に **EncryptionInProgress** が表示される場合、これは暗号化が進行中であることを意味します。数分待機した後に再試行します。

15.4. ETCD 暗号化の無効化

クラスターで etcd データの暗号化を無効にできます。

前提条件

- **cluster-admin** ロールを持つユーザーとしてクラスターにアクセスできる。

手順

1. **APIServer** オブジェクトを変更します。

```
$ oc edit apiserver
```

2. **encryption** フィールドタイプを **identity** に設定します。

```
spec:
  encryption:
    type: identity 1
```

1 **identity** タイプはデフォルト値であり、暗号化は実行されないことを意味します。

3. 変更を適用するためにファイルを保存します。
   復号化プロセスが開始されます。クラスターのサイズによっては、このプロセスが完了するまで 20 分以上かかる場合があります。

4. etcd の復号化が正常に行われたことを確認します。

   a. OpenShift API サーバーの **Encrypted** ステータス条件を確認し、そのリソースが正常に暗号化されたことを確認します。

   ```
   $ oc get openshiftapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

   ```
   DecryptionCompleted
   Encryption mode set to identity and everything is decrypted
   ```

   出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

   b. Kubernetes API サーバーの **Encrypted** ステータス状態を確認し、そのリソースが正常に復号化されたことを確認します。

   ```
   $ oc get kubeapiserver -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

   この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

   ```
   DecryptionCompleted
   Encryption mode set to identity and everything is decrypted
   ```

   出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

   c. OpenShift API サーバーの **Encrypted** ステータス条件を確認し、そのリソースが正常に復号化されたことを確認します。

   ```
   $ oc get authentication.operator.openshift.io -o=jsonpath='{range .items[0].status.conditions[?(@.type=="Encrypted")]}{.reason}{"\n"}{.message}{"\n"}'
   ```

この出力には、復号化が正常に実行されると **DecryptionCompleted** が表示されます。

```
DecryptionCompleted
Encryption mode set to identity and everything is decrypted
```

出力に **DecryptionInProgress** が表示される場合、これは復号化が進行中であることを意味します。数分待機した後に再試行します。

第16章 POD の脆弱性のスキャン

重要

Red Hat Quay Container Security Operator は非推奨となり、OpenShift Container Platform の今後のリリースで削除される予定です。Red Hat Quay Container Security Operator の公式置換製品は、Red Hat Advanced Cluster Security for Kubernetes です。

Red Hat Quay Container Security Operator を使用すると、OpenShift Container Platform Web コンソールから、クラスターのアクティブな Pod で使用されるコンテナーイメージに関する脆弱性スキャンの結果にアクセスできます。The Red Hat Quay Container Security Operator:

- すべての namespace または指定された namespace の Pod に関連付けられたコンテナーを監視します。
- イメージのレジストリーがイメージスキャンを実行している場合 (例: Quay.io、Clair スキャンを含む Red Hat Quay レジストリーなど)、脆弱性の情報についてコンテナーの出所となったコンテナーレジストリーをクエリーします。
- Kubernetes API の **ImageManifestVuln** オブジェクトを使用して脆弱性を公開します。

ここでの手順を使用すると、Red Hat Quay Container Security Operator は **openshift-operators** namespace にインストールされるため、OpenShift Container Platform クラスター上のすべての namespace で使用できます。

16.1. RED HAT QUAY CONTAINER SECURITY OPERATOR のインストール

Red Hat Quay Container Security Operator は、OpenShift Container Platform Web コンソール Operator Hub から、または CLI を使用してインストールできます。

前提条件

- **oc** CLI がインストールされている。
- OpenShift Container Platform クラスターへの管理者権限がある。
- クラスターで実行される Red Hat Quay または Quay.io レジストリーのコンテナーがある。

手順

1. OpenShift Container Platform Web コンソールを使用して Red Hat Quay Container Security Operator をインストールできます。
   a. Web コンソールで、**Operators → OperatorHub** に移動し、**Security** を選択します。
   b. **Red Hat Quay Container Security Operator**Operator を選択し、**Install** を選択します。
   c. **Red Hat Quay Container Security Operator**ページで、**Install** を選択します。**Update channel**、**Installation mode**、および **Update approval** が自動的に選択されます。**Installed Namespace** フィールドのデフォルトは **openshift-operators** です。必要に応じて、これらの設定を調整できます。

d. **Install** を選択します。しばらくすると、**Red Hat Quay Container Security Operator** は **Installed Operators** ページに表示されます。

e. オプション: カスタム証明書を Red Hat Quay Container Security Operator に追加できます。たとえば、現在のディレクトリーに **quay.crt** という名前の証明書を作成します。次に、以下のコマンドを実行してカスタム証明書を Red Hat Quay Container Security Operator に追加します。

```
$ oc create secret generic container-security-operator-extra-certs --from-file=quay.crt -n openshift-operators
```

f. オプション: カスタム証明書を追加した場合は、Red Hat Quay Container Security Operator Pod を再起動して、新しい証明書を有効にします。

2. あるいは、CLI を使用して Red Hat Quay Container Security Operator をインストールすることもできます。

a. 次のコマンドを入力して、Container Security Operator とそのチャネルの最新バージョンを取得します。

```
$ oc get packagemanifests container-security-operator \
  -o jsonpath='{range .status.channels[*]}{@.currentCSV} {@.name}{"\n"}{end}' \
  | awk '{print "STARTING_CSV=" $1 " CHANNEL=" $2 }' \
  | sort -Vr \
  | head -1
```

出力例

```
STARTING_CSV=container-security-operator.v3.8.9 CHANNEL=stable-3.8
```

b. 前のコマンドの出力を使用して、Red Hat Quay Container Security Operator の **Subscription** カスタムリソースを作成し、**container-security-operator.yaml** として保存します。以下に例を示します。

```
apiVersion: operators.coreos.com/v1alpha1
kind: Subscription
metadata:
  name: container-security-operator
  namespace: openshift-operators
spec:
  channel: ${CHANNEL} 1
  installPlanApproval: Automatic
  name: container-security-operator
  source: redhat-operators
  sourceNamespace: openshift-marketplace
  startingCSV: ${STARTING_CSV} 2
```

1 **spec.channel** パラメーターの直前の手順で取得した値を指定します。

2 **spec.startingCSV** パラメーターの直前の手順で取得した値を指定します。

c. 以下のコマンドを入力して設定を適用します。

```
$ oc apply -f container-security-operator.yaml
```

出力例

```
subscription.operators.coreos.com/container-security-operator created
```

16.2. RED HAT QUAY CONTAINER SECURITY OPERATOR の使用

以下の手順は、Red Hat Quay Container Security Operator を使用する方法を示しています。

前提条件

- Red Hat Quay Container Security Operator がインストールされている。

手順

1. OpenShift Container Platform Web コンソールで、**Home → Overview** に移動します。**Status** セクションの **Image Vulnerabilities** には、見つかった脆弱性の数が表示されます。
2. **Image Vulnerabilities** をクリックすると、**Image Vulnerabilities breakdown** タブが表示されます。このタブには、脆弱性の重大度、脆弱性を修正できるかどうか、脆弱性の総数などの詳細が表示されます。
3. 検出された脆弱性は、次の 2 つの方法のいずれかで対処できます。
    a. **Vulnerabilities** セクションのリンクを選択します。コンテナーを取得したコンテナーレジストリーにアクセスし、脆弱性に関する情報を確認できます。
    b. **namespace** リンクを選択します。これにより、**Image Manifest Vulnerabilities** ページに移動し、選択したイメージの名前と、そのイメージが実行しているすべての namespace を確認できます。
4. どのイメージに脆弱性があるか、それらの脆弱性を修正する方法、およびイメージが実行されている namespace を理解したら、次のアクションを実行することでセキュリティーを向上できます。
    a. 組織内のイメージを実行している人に警告し、脆弱性を修正するよう依頼してください。
    b. イメージが置かれている Pod を起動したデプロイメントまたは他のオブジェクトを削除して、イメージの実行を停止します。

注記

Pod を削除すると、ダッシュボードで脆弱性情報がリセットされるまで数分かかる場合があります。

16.3. CLI でのイメージ脆弱性のクエリー

oc コマンドを使用して、Red Hat Quay Container Security Operator によって検出される脆弱性に関する情報を表示できます。

前提条件

- Red Hat Quay Container Security Operator が OpenShift Container Platform インスタンスにインストールされている。

手順

1. を下のコマンドを実行して、検出されたコンテナーイメージの脆弱性をクエリーします。

```
$ oc get vuln --all-namespaces
```

出力例

```
NAMESPACE     NAME              AGE
default       sha256.ca90...    6m56s
skynet        sha256.ca90...    9m37s
```

2. 特定の脆弱性の詳細を表示するには、脆弱性の名前およびその namespace を **oc describe** コマンドに追加します。以下の例は、イメージに脆弱性のある RPM パッケージが含まれるアクティブなコンテナーを示しています。

```
$ oc describe vuln --namespace mynamespace sha256.ac50e3752...
```

出力例

```
Name:         sha256.ac50e3752...
Namespace:    quay-enterprise
...
Spec:
  Features:
    Name:            nss-util
    Namespace Name:  centos:7
    Version:         3.44.0-3.el7
    Versionformat:   rpm
    Vulnerabilities:
      Description: Network Security Services (NSS) is a set of libraries...
```

16.4. RED HAT QUAY CONTAINER SECURITY OPERATOR のアンインストール

Container Security Operator をアンインストールするには、Operator をアンインストールし、**imagemanifestvulns.secscan.quay.redhat.com** カスタムリソース定義 (CRD) を削除する必要があります。

手順

1. OpenShift Container Platform Web コンソールで、**Operators → Installed Operators** をクリックします。
2. Container Security Operator のメニュー をクリックします。
3. **Uninstall Operator** をクリックします。
4. ポップアップウィンドウで **Uninstall** をクリックして、決定を確認します。
5. CLI を使用して、**imagemanifestvulns.secscan.quay.redhat.com** CRD を削除します。

a. 次のコマンドを入力して、**imagemanifestvulns.secscan.quay.redhat.com** カスタムリソース定義を削除します。

```
$ oc delete customresourcedefinition imagemanifestvulns.secscan.quay.redhat.com
```

出力例

```
customresourcedefinition.apiextensions.k8s.io
"imagemanifestvulns.secscan.quay.redhat.com" deleted
```

第17章 NETWORK-BOUND DISK ENCRYPTION (NBDE)

17.1. ディスクの暗号化技術について

Network-Bound Disk Encryption(NBDE) を使用すると、マシンの再起動時にパスワードを手動で入力しなくても、物理マシンおよび仮想マシン上のハードドライブのルートボリュームを暗号化できます。

17.1.1. ディスク暗号化技術の比較

エッジサーバーにあるデータのセキュリティーを保護するための Network-Bound Disk Encryption (NBDE) のメリットを理解するには、キー Escrow と Clevis を使用しない TPM ディスク暗号化と、Red Hat Enterprise Linux (RHEL) を実行するシステム上の NBDE を比較します。

以下の表は、脅威モデルや各暗号化ソリューションの複雑さに関して考慮すべきトレードオフを示しています。

シナリオ	キーエスクロー	TPM ディスク暗号化 (Clevis なし)	NBDE
単一ディスクの盗難からの保護	X	X	X
サーバー全体に対する保護	X		X
システムはネットワークから独立して再起動できます。		X	
定期的なキーの再生成なし		X	
キーはネットワーク経由で送信されることはありません。		X	X
OpenShift にサポート		X	X

17.1.1.1. キーエスクロー

キーエスクローは、暗号鍵を保存するための従来のシステムです。ネットワーク上の鍵サーバーが、暗号化されたブートディスクを持つノードの暗号鍵を保存し、暗号鍵の照会時にそれを返します。キー管理、トランスポートの暗号化、および認証が複雑であるため、ブートディスク暗号化は妥当な選択肢ではありません。

Red Hat Enterprise Linux(RHEL) で利用可能ですが、主要な escrow ベースのディスク暗号化の設定および管理は手動のプロセスであり、ノードの自動追加など OpenShift Container Platform 自動化操作には適しておらず、現在 OpenShift Container Platform でサポートされていません。

17.1.1.2. TPM 暗号化

Trusted Platform Module (TPM) ディスク暗号化は、リモートから保護された場所のデータセンターまたはインストールに適しています。dm-crypt and BitLocker などの完全なディスク暗号化ユーティリティーは、TPM バインドキーでディスクを暗号化し、TPM に TPM バンドキーを保存し、ノードのマザーボードにアタッチします。この方法の主な利点は、外部の依存関係がなく、ノードは外部の対話なしにブート時に独自のディスクを復号化できることです。

TPM ディスク暗号化は、ディスクがノードから盗まれて外部で分析された場合でも、データの復号化を防ぎます。ただし、セキュアではない場所では、これは不十分です。たとえば、攻撃者がノード全体を盗んだ場合に、ノードは自身のディスクを復号化するので、ノードの電源を投入した時点でデータを傍受できてしまいます。これは、物理 TPM2 チップを備えたノードと、Virtual Trusted Platform Module(VTPM) アクセスのある仮想マシンに適用されます。

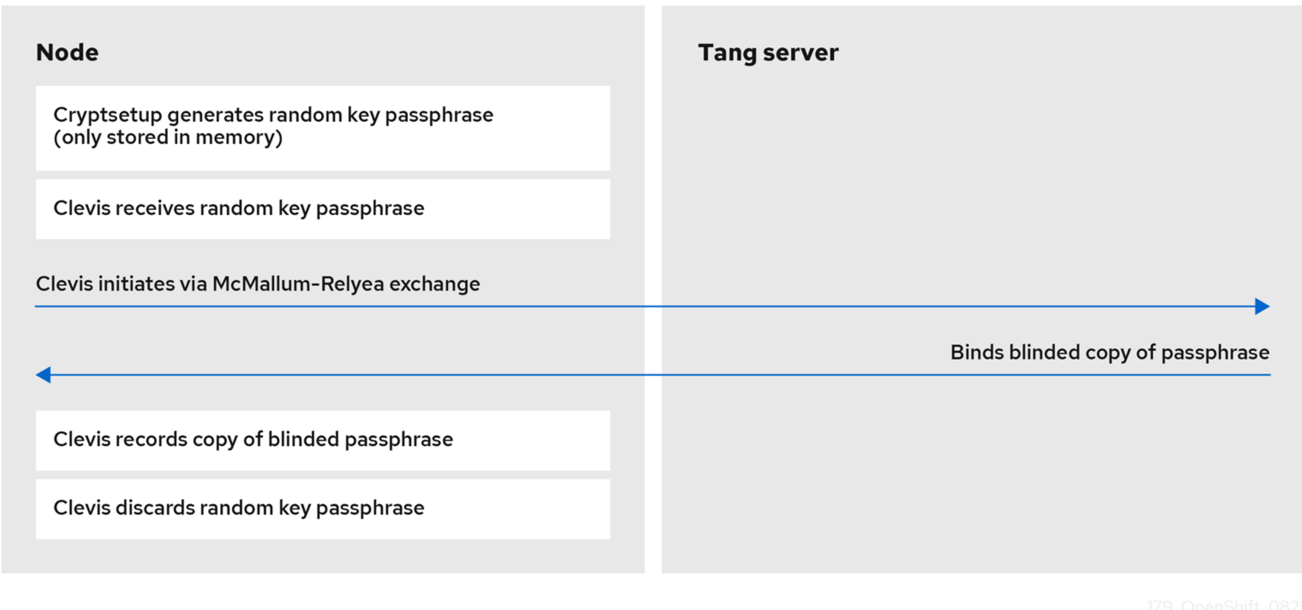
17.1.1.3. Network-Bound Disk Encryption (NBDE)

Network-Bound Disk Encryption (NBDE) は、ネットワーク全体でセキュアかつ匿名の方法で、暗号鍵を外部のサーバーまたはサーバーセットに効果的に関連付けます。これは、ノードが暗号鍵を保存したり、ネットワーク経由で転送したりしない点でキーエスクローと異なりますが、それ以外の点では同様に動作します。

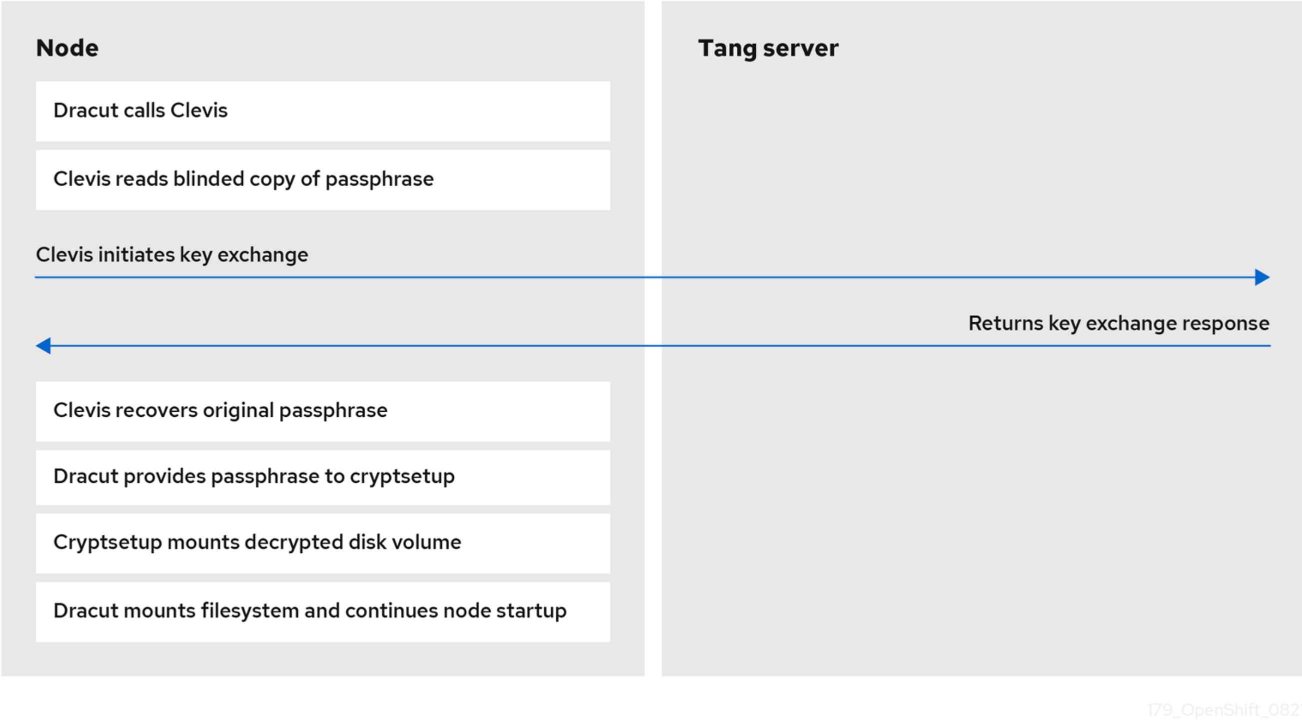
Clevis および Tang は、一般的なクライアントおよびサーバーのコンポーネントで、ネットワークがバインドされた暗号化を提供します。Red Hat Enterprise Linux CoreOS(RHCOS) は、Linux Unified Key Setup-on-disk-format(LUKS) と合わせて、これらのコンポーネントを使用して、Network-Bound Disk 暗号化を実現するために root および root 以外のストレージボリュームを暗号化および復号化します。

ノードが起動すると、暗号化ハンドシェイクを実行して事前に定義された Tang サーバーのセットへのアクセスを試みます。必要な数の Tang サーバーに到達できる場合は、ノードはディスクの復号化キーを作成し、ディスクのロックを解除して起動を続行できます。ネットワークが停止したり、サーバーが利用できなくなったりしたことが原因で、ノードが Tang サーバーにアクセスできない場合に、ノードは起動できず、Tang サーバーが再び利用可能になるまで無限に再試行し続けます。この鍵は実質的にはネットワーク内のノードに関連付けられるため、攻撃者が使用されていないデータへのアクセス権を取得しようとすると、ノードのディスクと、Tang サーバーへのネットワークアクセスを取得する必要があります。

以下の図は、NBDE のデプロイメントモデルを示しています。



以下の図は、リブート時の NBDE の動作を示しています。



17.1.1.4. シークレット共有の暗号化

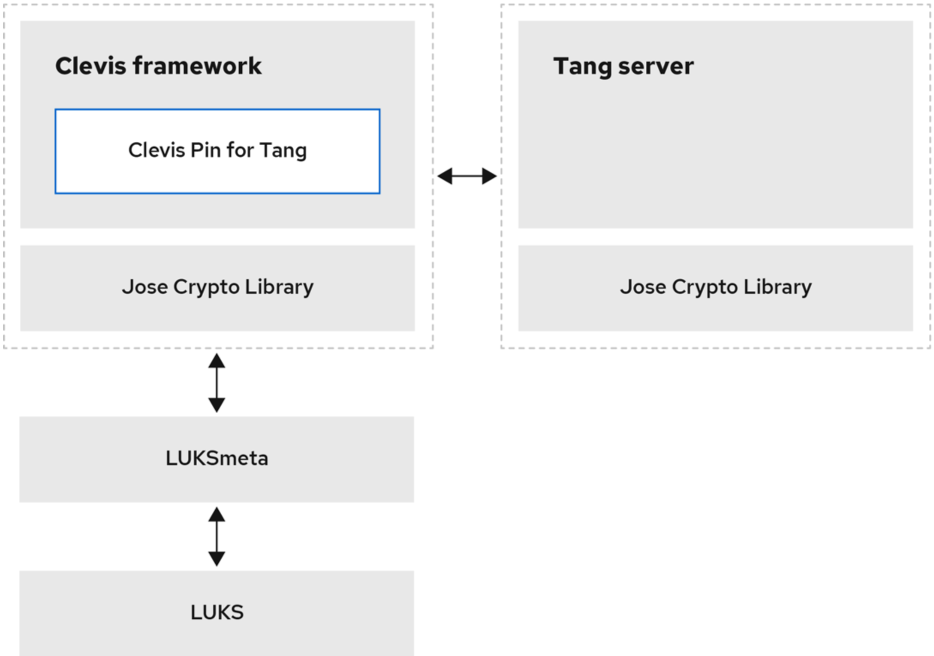
Shamir のシークレット共有 (sss) は、キーを安全に分割、配布、および再構築するための暗号化アルゴリズムです。このアルゴリズムを使用すると、OpenShift Container Platform は、より複雑なキー保護の組み合わせをサポートすることができます。

複数の Tang サーバーを使用するようにクラスターノードを設定する場合には、OpenShift Container Platform は、sss を使用して指定されたサーバーが 1 つ以上利用可能になると正常に実行される復号化ポリシーを設定します。追加でセキュリティー階層を作成できます。たとえば、OpenShift Container Platform でディスクの復号化に TPM と Tang サーバーの指定リストの 1 つを必要とするポリシーを定義できます。

17.1.2. Tang サーバーディスク暗号化

以下のコンポーネントおよびテクノロジーは、Network-Bound Disk Encryption(NBDE) を実装します。

図17.1 LUKS1 で暗号化したボリュームを使用する場合の NBDE スキーム(luksmeta パッケージは、LUKS2 ボリュームには使用されません)



Tang は、ネットワークのプレゼンスにデータをバインドするためのサーバーです。これにより、ノードが特定のセキュアなネットワークにバインドされると、データが含まれるノードが利用可能になります。Tang はステートレスであり、Transport Layer Security(TLS) または認証は必要ありません。鍵サーバーがすべての暗号鍵を保存し、すべての暗号鍵に関する情報を有するエスクローベースのソリューションと異なり、Tang はノードの鍵とやり取りしないため、ノードから識別情報を取得することがありません。

Clevis は、自動復号化用のプラグ可能なフレームワークで、Linux Unified Key Setup-on-disk-format(LUKS) ボリュームの自動ロック解除機能が含まれます。Clevis パッケージはノードで実行され、クライアント側の機能を提供します。

Clevis ピン は、Clevis フレームワークへのプラグインです。ピニングには、以下の 3 つのタイプがあります。

TPM2

ディスク暗号化を TPM2 にバインドします。

Tang

ディスク暗号化を Tang サーバーにバインドし、NBDE を有効にします。

Shamir のシークレット共有 (sss)

より複雑な他のピニングの組み合わせを可能にします。これにより、以下のような冗長なポリシーが可能になります。

- これらの 3 つの Tang サーバーのいずれかに到達できる必要があります。
- これら 5 つの Tang サーバーのうち 3 つに到達できる必要があります
- TPM2 と、これらの 3 つの Tang サーバーのいずれかに到達できる必要があります。

17.1.3. Tang サーバーロケーションのプランニング

Tang サーバー環境を計画する時には、Tang サーバーの物理ネットワークおよびネットワークの場所を検討してください。

物理的な場所

Tang サーバーの地理的な場所は、不正アクセスや盗難から適切に保護されており、重要なサービスを実行するために必要な可用性があり、アクセスできる場合には、比較的重要ではありません。Clevis クライアントを使用するノードには、Tang サーバーが常に利用可能である限り、ローカルの Tang サーバーは必要ありません。障害復旧には、その場所に関係なく、Tang サーバーに対して電源とネットワーク接続がいずれも冗長設定されている必要があります。

ネットワークの場所

Tang サーバーにネットワークアクセスできるノードは、独自のディスクパーティション、または同じ Tang サーバーで暗号化されたその他のディスクを復号化できます。
Tang サーバーのネットワークの場所を選択し、指定のホストからネットワーク接続の有無によって復号化のパーミッションが許可されるようにします。たとえば、ファイアウォール保護を適用して、あらゆるタイプのゲストネットワークやパブリックネットワーク、またはセキュリティーで保護されていない建物のエリアにあるネットワークジャックからのアクセスを禁止できます。

また、実稼働ネットワークと開発ネットワークの間でネットワークを分離します。これにより、適切なネットワークの場所を定義し、セキュリティーの層を追加するのに役立ちます。

ロック解除を行う Tang サーバーを同じリソース (例: 同じ **rolebindings.rbac.authorization.k8s.io** クラスター) にデプロイしないでください。ただし、Tang サーバーおよび他のセキュリティーリソースのクラスターは、複数の追加クラスターおよびクラスターリソースのサポートを有効にする、便利な設定である場合があります。

17.1.4. Tang サーバーのサイジング要件

可用性、ネットワーク、および物理的な場所に関する要件をもとに、サーバー容量の懸念点が決まるのではなく、使用する Tang サーバー数を決定できます。

Tang サーバーは、Tang リソースを使用して暗号化されたデータの状態を維持しません。Tang サーバーは完全に独立しているか、キー情報のみを共有するので、スケーリングが可能です。

以下の 2 つの方法を使用して、Tang サーバーで重要な情報を処理できます。

- 複数の Tang サーバーが重要な情報を共有します。
  - 同じ URL の背後にあるキーを共有する Tang サーバーを負荷分散する必要があります。この設定はラウンドロビン DNS と同様に簡単に実行することも、物理ロードバランサーを使用することもできます。
  - 単一の Tang サーバーから複数の Tang サーバーにスケーリングできます。Tang サーバーのスケーリングでは、Tang サーバーがキー情報と同じ URL を共有する場合にノードのキー変更やクライアントの再設定は必要ありません。
  - クライアントノードの設定とキーローテーションには Tang サーバー 1 台のみが必要です。
- 複数の Tang サーバーは独自のキー情報を生成します。
  - インストール時に複数の Tang サーバーを設定できます。
  - ロードバランサーの背後では個別の Tang サーバーをスケーリングできます。

- クライアントノードの設定またはキーのローテーション時に全 Tang サーバーが利用可能である必要があります。
- クライアントノードがデフォルト設定を使用して起動すると、Clevis クライアントはすべての Tang サーバーに接続します。復号化を続行するには、**n** 台の Tang サーバーのみがオンラインになっている必要があります。**n** のデフォルト値は 1 です。
- Red Hat では、Tang サーバーの動作を変更するインストール後の設定をサポートしません。

17.1.5. ロギングに関する考慮事項

Tang トラフィックの集中ロギングは、予期しない復号化要求などを検出できる可能性があるため、便利です。以下に例を示します。

- ブートシーケンスに対応しないパスフレーズの復号化を要求するノード
- 既知のメンテナンスアクティビティー外の復号化を要求するノード (例: 繰り返しキー)

17.2. TANG サーバーのインストールに関する考慮事項

Network-Bound Disk Encryption(NBDE) はクラスターノードのインストール時に有効にする必要があります。ただし、ディスク暗号化ポリシーは、インストール時の初期化後に、いつでも変更できます。

17.2.1. インストールシナリオ

Tang サーバーインストールを計画する場合は、以下の推奨事項を検討してください。

- 小規模な環境では、複数の Tang サーバーを使用する場合でも、単一のキー情報のセットを使用できます。
  - キーのローテーションが容易になりました。
  - Tang サーバーは、高可用性を確保できるように簡単にスケーリングできます。
- 大規模な環境には、複数のキー情報のセットからメリットを得ることができます。
  - 物理的なインストールでは、地理的リージョン間のキー情報のコピーおよび同期は必要ありません。
  - キーローテーションは大規模な環境ではより複雑です。
  - ノードのインストールおよびキー変更には、すべての Tang サーバーへのネットワーク接続が必要です。
  - ブートノードが復号化中にすべての Tang サーバーに対してクエリーを実行するため、ネットワークトラフィックがわずかに増加する可能性があります。成功する必要があるのは Clevis クライアントクエリー 1 つのみですが、Clevis はすべての Tang サーバーにクエリーを実行することに注意してください。
- 複雑性:
  - 追加の手動再設定では、ディスクパーティションを復号化するために、**オンラインのサーバー M 台中 N 台** の Shamir シークレット共有 (sss) を許可できます。このシナリオでディスクを復号化するには、複数のキー情報のセットと、初回インストール後に Clevis クライアントが含まれる Tang サーバーとノードを手動で管理する必要があります。

- ハイレベルの推奨事項:
    - 単一の RAN デプロイメントの場合には、対応するドメインコントローラー (DC) で、一部の Tang サーバーセットを実行できます。
    - 複数の RAN デプロイメントの場合には、対応する各 DC で Tang サーバーを実行するか、グローバル Tang 環境が他のシステムのニーズと要件に適しているかどうかを決定する必要があります。

17.2.2. Tang サーバーのインストール

1 つ以上の Tang サーバーをデプロイするには、シナリオに応じて次のオプションから選択します。

1. NBDE Tang Server Operator を使用して Tang サーバーをデプロイする
2. enforcing モードの SELinux を使用して Tang サーバーをデプロイする
3. RHEL Web コンソールで Tang サーバーを設定する
4. コンテナーとしての Tang のデプロイ
5. 複数の Tang サーバーのセットアップに nbde_server システムロールを使用する

17.2.2.1. コンピュートの要件

Tang サーバーの計算要件は非常に低くなります。サーバーを実稼働環境にデプロイするのに使用する通常のサーバーグレードの設定であれば、十分なコンピュート容量をプロビジョニングできます。

高可用性に関する考慮事項は、可用性のみを対象としており、クライアントの要件に対応するための追加のコンピュート能力ではありません。

17.2.2.2. 起動時の自動開始

Tang サーバーが使用するキー情報は機密性が高いため、Tang サーバーの起動シーケンス中の手動介入のオーバーヘッドが有益である可能性があるので、忘れないでください。

デフォルトでは、Tang サーバーが起動し、想定されるローカルボリュームにキー情報がない場合には、新しいマテリアルを作成し、それを提供します。既存のキー情報を使用して開始するか、起動を中止して手動の介入を待つことで、このデフォルトの動作を避けることができます。

17.2.2.3. HTTP 対 HTTPS

Tang サーバーへのトラフィックは、暗号化 (HTTPS) することも、プレーンテキスト (HTTP) にすることもできます。このトラフィックを暗号化してもセキュリティーの面で大きな利点はなく、トラフィックを復号化したままにすると、Clevis クライアントを実行するノードでのトランスポート層セキュリティー (TLS) 証明書チェックに関連する複雑性や障害状態がなくなります。

ノードの Clevis クライアントと Tang サーバー間の暗号化されていないトラフィックのパッシブモニタリングを実行することは可能ですが、このトラフィックを使用してキーの情報を判断する機能はせいぜい、理論的な懸念事項を判断できる程度です。このようなトラフィック分析には、大量のキャプチャーデータが必要になります。キーローテーションはすぐに無効になります。最後に、パッシブモニタリングを実行できる脅威アクターは、Tang サーバーへの手動接続に必要なネットワークアクセスをすでに取得しており、キャプチャーされた Clevis ヘッダーの単純な復号化を実行できます。

ただし、インストールサイトで実施されている他のネットワークポリシーでは、アプリケーションに関係なくトラフィックの暗号化が必要になる場合があるため、この決定はクラスター管理者に任せることを検討してください。

関連情報

- RHEL 8 セキュリティーハードニング ドキュメントの ポリシーベースの復号化を使用した暗号化ボリュームの自動ロック解除の設定
- Tang の正式なサーバーコンテナー
- インストール時のディスクの暗号化およびミラーリング

17.3. TANG サーバーの暗号化キー管理

暗号鍵を再作成するための暗号化メカニズムは、ノードに保存されている **ブラインドキー** と、関連する Tang サーバーの秘密鍵に基づいています。

> **注記**
>
> Tang サーバーの秘密鍵とノードの暗号化ディスクの両方を取得した者による攻撃から保護するには、定期的にキーを再生成することを推奨します。
>
> Tang サーバーから古いキーを削除する前に、すべてのノードでキー変更操作を実行する必要があります。

以下のセクションでは、古いキーを再生成し、削除する手順を説明します。

17.3.1. Tang サーバーのキーのバックアップ

Tang サーバーは **/usr/libexec/tangd-keygen** を使用して新しいキーを生成し、デフォルトで **/var/db/tang** ディレクトリーに保存します。障害が発生した場合に Tang サーバーを回復するには、このディレクトリーをバックアップします。キーは機密性が高く、使用した全ホストのブートディスクを復号化できるため、キーは適切に保護される必要があります。

手順

- バックアップキーを **/var/db/tang** ディレクトリーから、キーを復元できる temp ディレクトリーにコピーします。

17.3.2. Tang サーバーのキーのリカバリー

バックアップからキーにアクセスして、Tang サーバーのキーを回復できます。

手順

- キーをバックアップフォルダーから **/var/db/tang/** ディレクトリーに復元します。
  Tang サーバーの起動時に、これらの復元されたキーをアドバタイズして使用します。

17.3.3. Tang サーバーのキー変更

以下の手順では、例として一意のキーが割り当てられた 3 つの Tang サーバーセットを使用します。

冗長な Tang サーバーを使用すると、ノードの自動起動に失敗する可能性が低減します。

Tang サーバーおよび関連付けられたすべての NBDE 暗号化ノードのキーは 3 つの手順を使用して、再生成します。

前提条件

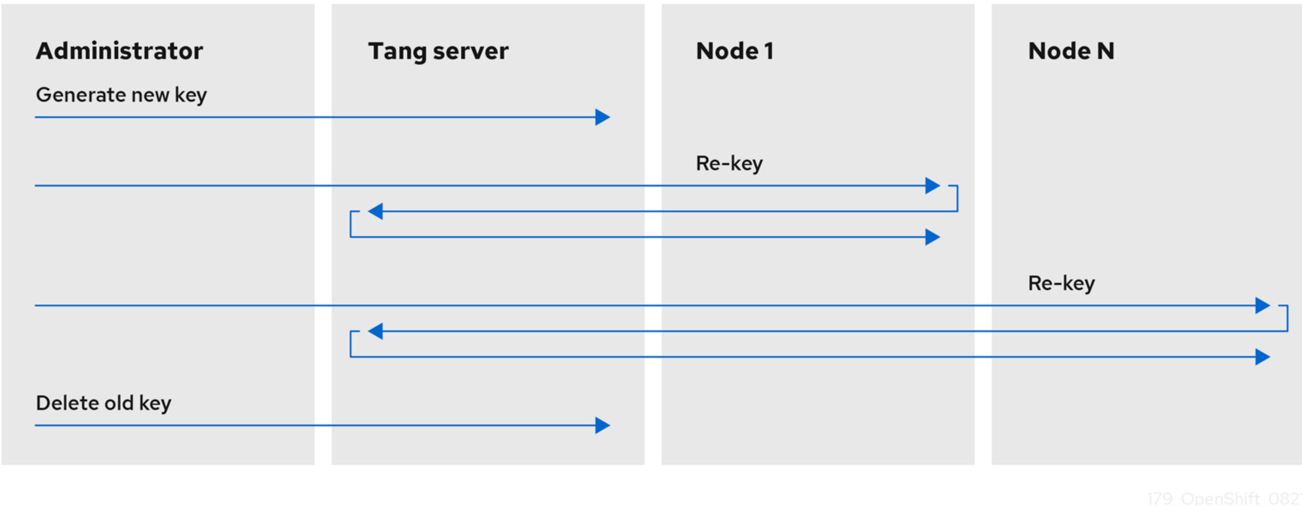
- 1 つ以上のノードで機能する Network-Bound Disk Encryption(NBDE) をインストールしておく。

手順

1. 新しい Tang サーバーキーを生成します。
2. NBDE で暗号化された全ノードに新規キーを使用するようにキーを再生成します。
3. 古い Tang サーバーキーを削除します。

> **注記**
>
> NBDE で暗号化されたすべてのノードがキーの再生成を完了する前に古いキーを削除すると、それらのノードは他の設定済みの Tang サーバーに過度に依存するようになります。

図17.2 Tang サーバーのキー再生成のワークフロー例



17.3.3.1. 新しい Tang サーバーキーの生成

前提条件

- Tang サーバーを実行する Linux マシンのルートシェル。
- Tang サーバーキーのローテーションを容易に検証するには、古いキーで小規模なテストファイルを暗号化します。

```
# echo plaintext | clevis encrypt tang '{"url":"http://localhost:7500"}' -y >/tmp/encrypted.oldkey
```

- 暗号化が正常に完了し、ファイルを復号化して同じ文字列 **plaintext** を生成できることを確認します。

```
# clevis decrypt </tmp/encrypted.oldkey
```

手順

1. Tang サーバーキーの保存先のディレクトリーを見つけてアクセスします。通常、これは **/var/db/tang** ディレクトリーです。現在公開されているキーのサムプリントを確認します。

```
# tang-show-keys 7500
```

出力例

```
36AHjNH3NZDSnlONLz1-V4ie6t8
```

2. Tang サーバーのキーディレクトリーを入力します。

```
# cd /var/db/tang/
```

3. 現在の Tang サーバーキーをリスト表示します。

```
# ls -A1
```

出力例

```
36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
```

通常の Tang サーバーの操作時に、このディレクトリーには、署名および検証用とキー派生用の 2 つの **.jwk** ファイルがあります。

4. 古いキーのアドバタイズを無効にします。

```
# for key in *.jwk; do \
  mv -- "$key" ".$key"; \
done
```

Network-Bound Disk Encryption(NBDE) を使用する新規クライアント、またはキーを要求する新規クライアントには古いキーは表示されなくなりました。既存のクライアントは、削除されるまで以前のキーにアクセスして使用できます。Tang サーバーは、**.** 文字で始まる、UNIX の非表示ファイルに保存されているキーを読み取りますがアドバタイズはしません。

5. 新しいキーを生成します。

```
# /usr/libexec/tangd-keygen /var/db/tang
```

6. これらのファイルは非表示になり、新しいキーが存在するので、現在の Tang サーバーキーをリスト表示して古いキーがアドバタイズされなくなったことを確認します。

```
# ls -A1
```

出力例

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

Tang は、新しいキーを自動的にアドバタイズします。

> **注記**
>
> より新しい Tang サーバーのインストールには、アドバタイズを無効にして新規キーを同時に生成するヘルパー **/usr/libexec/tangd-rotate-keys** ディレクトリーが含まれます。

7. 同じキー情報を共有するロードバランサーの背後で複数の Tang サーバーを実行している場合は、続行する前に、ここで加えた変更が一連のサーバー全体に適切に同期されていることを確認します。

検証

1. Tang サーバーが、古いキーではなく、新しいキーをアドバタイズすることを確認します。

```
# tang-show-keys 7500
```

出力例

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. アドバタイズされていない古いキーがまだ復号化要求に使用できることを確認します。

```
# clevis decrypt </tmp/encrypted.oldkey
```

17.3.3.2. 全 NBDE ノードのキー変更

リモートクラスターにダウンタイムを発生させずに **DaemonSet** オブジェクトを使用することで、リモートクラスターのすべてのノードにキーを再生成できます。

> **注記**
>
> キー設定時にノードの電源が切れると、起動できなくなる可能性があり、Red Hat Advanced Cluster Management(RHACM) または GitOps パイプラインを使用して再デプロイする必要があります。

前提条件

- Network-Bound Disk Encryption(NBDE) ノードが割り当てられたすべてのクラスターへの **cluster-admin** アクセス。
- Tang サーバーのキーが変更されていない場合でも、キーの再生成が行われるすべての NBDE ノードからすべての Tang サーバーにアクセスできる必要があります。

- すべての Tang サーバーの Tang サーバーの URL およびキーのサムプリントを取得します。

手順

1. 以下のテンプレートに基づいて **DaemonSet** オブジェクトを作成します。このテンプレートは 3 つの冗長 Tang サーバーを設定しますが、他の状況にも簡単に対応できます。**NEW_TANG_PIN** 環境の Tang サーバーの URL およびサムプリントを、実際の環境に合わせて変更します。

```
apiVersion: apps/v1
kind: DaemonSet
metadata:
  name: tang-rekey
  namespace: openshift-machine-config-operator
spec:
  selector:
    matchLabels:
      name: tang-rekey
  template:
    metadata:
      labels:
        name: tang-rekey
    spec:
      containers:
      - name: tang-rekey
        image: registry.access.redhat.com/ubi9/ubi-minimal:latest
        imagePullPolicy: IfNotPresent
        command:
        - "/sbin/chroot"
        - "/host"
        - "/bin/bash"
        - "-ec"
        args:
        - |
          rm -f /tmp/rekey-complete || true
          echo "Current tang pin:"
          clevis-luks-list -d $ROOT_DEV -s 1
          echo "Applying new tang pin: $NEW_TANG_PIN"
          clevis-luks-edit -f -d $ROOT_DEV -s 1 -c "$NEW_TANG_PIN"
          echo "Pin applied successfully"
          touch /tmp/rekey-complete
          sleep infinity
        readinessProbe:
          exec:
            command:
            - cat
            - /host/tmp/rekey-complete
          initialDelaySeconds: 30
          periodSeconds: 10
        env:
        - name: ROOT_DEV
          value: /dev/disk/by-partlabel/root
        - name: NEW_TANG_PIN
          value: >-
            {"t":1,"pins":{"tang":[
              {"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
```

```
          {"url":"http://tangserver02:7500","thp":"I5Ynh2JefoAO3tNH9TgI4obIaXI"},
          {"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
        ]}}
    volumeMounts:
    - name: hostroot
      mountPath: /host
    securityContext:
      privileged: true
  volumes:
  - name: hostroot
    hostPath:
      path: /
  nodeSelector:
    kubernetes.io/os: linux
  priorityClassName: system-node-critical
  restartPolicy: Always
  serviceAccount: machine-config-daemon
  serviceAccountName: machine-config-daemon
```

この場合は、**tangserver01** のキーを再生成していても、**tangserver01** の新規サムプリントだけでなく、その他すべての Tang サーバーの現在のサムプリントも指定する必要があります。キーの再生成操作にすべてのサムプリントの指定に失敗すると、中間者攻撃の可能性が高まります。

2. キーの再生成が必要なすべてのクラスターにデーモンセットを配布するには、次のコマンドを実行します。

```
$ oc apply -f tang-rekey.yaml
```

ただし、スケーリングで実行するには、デーモンセットを ACM ポリシーでラップします。この ACM 設定には、デーモンセットをデプロイするポリシーが 1 つ、すべてのデーモンセット Pod が READY であることを確認する 2 番目のポリシー、およびクラスターの適切なセットに適用する配置ルールを含める必要があります。

注記

デーモンセットのすべてのサーバーが正常に再割り当てされたことを確認したら、デーモンセットを削除します。デーモンセットを削除しない場合は、次回のキー再生成操作の前にこれを削除する必要があります。

検証

デーモンセットを配布したら、デーモンセットを監視し、キー作成が正常に完了したことを確認します。サンプルのデーモンセットのスクリプトは、キー変更に失敗するとエラーで終了し、成功した場合には **CURRENT** 状態のままになります。また、キーの再生成が正常に実行されると、Pod に **READY** のマークを付ける readiness プローブもあります。

- 以下は、キー変更が完了する前に設定されたデーモンセットの出力リストの例です。

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

出力例

```
NAME        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR          AGE
tang-rekey   1         1         0       1           0          kubernetes.io/os=linux   11s
```

- 以下は、キーの変更が正常に実行された後のデーモンセットの出力リストの例です。

```
$ oc get -n openshift-machine-config-operator ds tang-rekey
```

出力例

```
NAME        DESIRED   CURRENT   READY   UP-TO-DATE   AVAILABLE   NODE
SELECTOR          AGE
tang-rekey   1         1         1       1           1          kubernetes.io/os=linux   13h
```

キーの再生成には、通常完了までに数分かかります。

注記

ACM ポリシーを使用してデーモンセットを複数のクラスターに分散する場合には、すべてのデーモンセットの READY の数が DESIRED の数と等しいことを確認するコンプライアンスポリシーを含める必要があります。こうすることで、このようなポリシーに準拠すると、すべてのデーモンセット Pod が READY で、キーの再生成が正常に実行されていることが分かります。ACM 検索を使用して、デーモンセットのすべての状態をクエリーすることもできます。

17.3.3.3. Tang サーバーの一時的な再生成エラーのトラブルシューティング

Tang サーバーのキーの再生成のエラー状態が一時的かどうかを判別するには、以下の手順を実行します。一時的なエラー状態には以下が含まれます。

- 一時的なネットワークの停止
- Tang サーバーのメンテナンス

通常、これらのタイプの一時的なエラー状態が発生した場合には、デーモンセットがエラーを解決して成功するまで待機するか、デーモンセットを削除し、一時的なエラー状態が解決されるまで再試行しないようにします。

手順

1. 通常の Kubernetes Pod 再起動ポリシーを使用してキーの再生成操作を実行する Pod を再起動します。
2. 関連付けられた Tang サーバーのいずれかが利用できない場合は、すべてのサーバーがオンラインに戻るまでキーの再生成を試みます。

17.3.3.4. Tang サーバーの永続的な再生成エラーのトラブルシューティング

Tang サーバーのキーを再生成した後に、長期間経過しても **READY** の数が **DESIRED** の数と等しくない場合は、永続的な障害状態を示している可能性があります。この場合、以下のような状況である場合があります。

- **NEW_TANG_PIN** 定義で Tang サーバーの URL またはサムプリントの誤字がある。

- Tang サーバーが無効になっているか、キーが完全に失われている。

前提条件

- この手順で説明されているコマンドが、Tang サーバーまたは Tang サーバーへのネットワークアクセスのある Linux システムで実行できる。

手順

1. デーモンセットの定義に従って各 Tang サーバーの設定に対して単純な暗号化および復号化操作を実行して、Tang サーバー設定を検証します。
   これは、不適切なサムプリントで暗号化および復号化を試行する例です。

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver02:7500","thp":"badthumbprint"}' | \
  clevis decrypt
```

   出力例

```
Unable to fetch advertisement: 'http://tangserver02:7500/adv/badthumbprint'!
```

   これは、正常なサムプリントで暗号化および復号化を試行する例です。

```
$ echo "okay" | clevis encrypt tang \
  '{"url":"http://tangserver03:7500","thp":"goodthumbprint"}' | \
  clevis decrypt
```

   出力例

```
okay
```

2. 根本的な原因を特定した後に、その原因となる状況に対応します。
   a. 機能しないデーモンセットを削除します。
   b. デーモンセットの定義を編集して基礎となる問題を修正します。これには、以下のアクションのいずれかが含まれる場合があります。
      - Tang サーバーのエントリーを編集して URL とサムプリントを修正します。
      - 使用されなくなった Tang サーバーを削除します。
      - 使用停止したサーバーの代わりとなる、新規の Tang サーバーを追加します。
3. 更新されたデーモンセットを再度配布します。

注記

Tang サーバーを設定から置き換え、削除、または追加する場合に、現在キーの再生成が行われているサーバーを含め、少なくとも 1 つの元のサーバーが機能している限り、キーの再生成操作は成功します。元の Tang サーバーのいずれも機能しなくなるか、復元できない場合には、システムの回復は不可能で、影響を受けるノードを再デプロイする必要があります。

検証

デーモンセットの各 Pod からのログをチェックして、キーの再生成が正常に完了したかどうかを判断します。キーの再生成に成功しなかった場合には、その失敗している状態がログに表示される可能性があります。

1. デーモンセットによって作成されたコンテナーの名前を見つけます。

```
$ oc get pods -A | grep tang-rekey
```

出力例

```
openshift-machine-config-operator  tang-rekey-7ks6h  1/1  Running   20 (8m39s ago)  89m
```

2. コンテナーからログを印刷します。キーの再生成操作に成功すると、以下のログのようになります。

```
$ oc logs tang-rekey-7ks6h
```

出力例

```
Current tang pin:
1: sss '{"t":1,"pins":{"tang":[{"url":"http://10.46.55.192:7500"},{"url":"http://10.46.55.192:7501"},
{"url":"http://10.46.55.192:7502"}]}}'
Applying new tang pin: {"t":1,"pins":{"tang":[
  {"url":"http://tangserver01:7500","thp":"WOjQYkyK7DxY_T5pMncMO5w0f6E"},
  {"url":"http://tangserver02:7500","thp":"I5Ynh2JefoAO3tNH9TgI4obIaXI"},
  {"url":"http://tangserver03:7500","thp":"38qWZVeDKzCPG9pHLqKzs6k1ons"}
]}}
Updating binding...
Binding edited successfully
Pin applied successfully
```

17.3.4. 古い Tang サーバーキーの削除

前提条件

- Tang サーバーを実行する Linux マシンのルートシェル。

手順

1. Tang サーバーキーが保存されるディレクトリーを見つけ、これにアクセスします。通常、これは **/var/db/tang** ディレクトリーです。

```
# cd /var/db/tang/
```

2. 現在の Tang サーバーキーをリスト表示し、アドバタイズされたキーと、されていないキーを表示します。

```
# ls -A1
```

出力例

```
.36AHjNH3NZDSnlONLz1-V4ie6t8.jwk
.gJZiNPMLRBnyo_ZKfK4_5SrnHYo.jwk
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

3. 古いキーを削除します。

```
# rm .*.jwk
```

4. 現在の Tang サーバーのキーをリスト表示し、アドバタイズされていないキーが存在しなくなったことを確認します。

```
# ls -A1
```

出力例

```
Bp8XjITceWSN_7XFfW7WfJDTomE.jwk
WOjQYkyK7DxY_T5pMncMO5w0f6E.jwk
```

検証

この時点では、サーバーは新しいキーを引き続きアドバタイズしますが、古いキーをもとに復号化の試行は失敗します。

1. Tang サーバーに、現在公開されているキーのサムプリントをクエリーします。

```
# tang-show-keys 7500
```

出力例

```
WOjQYkyK7DxY_T5pMncMO5w0f6E
```

2. 先に作成したテストファイルを復号化して、以前のキーに対して復号化を検証します。

```
# clevis decrypt </tmp/encryptValidation
```

出力例

```
Error communicating with the server!
```

同じキー情報を共有するロードバランサーの背後に複数の Tang サーバーを実行している場合は、続行する前に、一連のサーバーで変更が適切に同期されていることを確認します。

17.4. 障害復旧に関する考慮事項

このセクションでは、発生する可能性のある障害状況と、それぞれの状況に対応する手順を説明します。その他の状況は、検出または想定される可能性の高い状況として追加されます。

17.4.1. クライアントマシンの損失

Tang サーバーを使用してディスクパーティションを復号化するクラスターノードが失われた場合は、**障害ではありません**。マシンの盗難、ハードウェアの障害、別の損失のシナリオが発生したのかは重要ではありません。ディスクは暗号化されて、回復不能とみなされます。

ただし、盗難が発生した場合は、Tang サーバーのキーを予防的にローテーションし、残りのすべてのノードのキーを再設定して、後で Tang サーバーに不正アクセスできた場合に備え、ディスクが回復不能なままにしておくことが懸命です。

この状況から回復するには、ノードを再インストールするか、置き換えます。

17.4.2. クライアントネットワーク接続が失われた場合のプランニング

個々のノードに対してネットワーク接続が失われると、無人で起動できなくなります。

ネットワーク接続が失われる可能性のある作業を計画している場合には、オンサイトの技術者に、手動で使用するパスフレーズを公開し、その後にキーを無効にしてローテーションすることができます。

手順

1. ネットワークが利用できなくなる前に、以下のコマンドでデバイス **/dev/vda2** の最初のスロット **-s 1** で使用されているパスワードを表示します。

```
$ sudo clevis luks pass -d /dev/vda2 -s 1
```

2. 以下のコマンドでその値を無効にし、新しい起動時のパスフレーズを無作為に再生成します。

```
$ sudo clevis luks regen -d /dev/vda2 -s 1
```

17.4.3. ネットワーク接続の予期しない損失

ネットワークが予期せず中断され、ノードが再起動する場合には、以下のシナリオを検討してください。

- いずれかのノードがまだオンラインのままである場合は、ネットワーク接続が復元されるまで再起動しないようにしてください。これは単一ノードクラスターには適用されません。
- ノードは、ネットワーク接続が復元されるまで、またはコンソールで設定したパスフレーズを手動で入力するまでノードはオフラインのままになります。このような状況では、ネットワーク管理者はアクセスを再確立するためにネットワークセグメントを再設定できる可能性がありますが、これは NBDE の意図に反します。つまり、ネットワークアクセスがないと起動できなくなります。
- ノードへのネットワークアクセスがないと、ノードの機能およびブート機能に影響を与えることが予想されます。ノードが手動の介入で起動されたとしても、ネットワークアクセスがないため、ノードは事実上役に立たなくなります。

17.4.4. ネットワーク接続の手動による回復

オンサイト技術者は、ネットワーク回復のために、やや複雑で手動での作業を多用するプロセスも利用できます。

手順

1. オンサイト技術者は、ハードディスクから Clevis ヘッダーを抽出します。BIOS のロックダウンによっては、ディスクを削除してラボマシンにインストールする必要がある場合があります。

2. オンサイトの技術者は、Tang ネットワークへの正当なアクセス権があるスタッフに Clevis ヘッダーを送信し、そのスタッフが復号化を実行します。

3. Tang ネットワークへのアクセスを制限するる必要があるため、技術者は VPN または他のリモート接続経由でそのネットワークにアクセス不可にする必要があります。同様に、技術者はこのネットワーク経由でリモートサーバーにパッチを適用して、ディスクを自動的に復号化できません。

4. ファイルシステムはディスクを再インストールし、それらが提供するプレーンテキストのパスフレーズを手動で入力します。

5. マシンは、Tang サーバーに直接アクセスしなくても、正常に起動します。インストールサイトからネットワークアクセスのある別のサイトにキー情報を転送する場合は注意して行ってください。

6. ネットワーク接続が回復すると、暗号鍵がローテーションされます。

17.4.5. ネットワーク接続の緊急復旧

ネットワーク接続を手動で回復できない場合には、以下の手順を検討してください。ネットワーク接続を回復する他の方法が利用できる場合には、これらの手順は推奨されないことに注意してください。

- この方法は、信頼性の高い技術者のみが実行する必要があります。

- Tang サーバーキー鍵情報をリモートサイトに送付することは、キー情報の違反とみなされ、すべてのサーバーでキーをもう一度生成して再暗号化する必要があります。

- この方法は、それ以外方法がない究極の場合に利用するようにしてください。あるいは、その実行可能性を実証するための概念実証の回復方法として使用する必要があります。

- 同様に極端で、理論的には可能ですが、問題のサーバーに無停電電源装置 (UPS) で電力を供給し、サーバーをネットワーク接続のある場所に転送してディスクを起動および復号化し、サーバーをバッテリー電源のある元の場所に復元して操作を続行します。

- バックアップの手動パスフレーズを使用する場合は、障害が発生する前にこれを作成する必要があります。

- 攻撃シナリオが TPM と Tang とスタンドアロンの Tang インストールと比較すると、より複雑になるので、同じ方法を使用する場合には、障害復旧プロセスも複雑になります。

17.4.6. ネットワークセグメントの喪失

ネットワークセグメントが失われ、Tang サーバーが一時的に使用できなくなると、次のような結果になります。

- OpenShift Container Platform ノードは、他のサーバーが利用可能な場合に通常通りに起動を続けます。

- 新規ノードは、ネットワークセグメントが復元されるまでその暗号化キーを確立できません。この場合は、高可用性および冗長性を確保するために、地理的に離れた場所への接続を確保します。これは、新規ノードのインストールや既存ノードの再割り当て時に、その操作で参照している Tang サーバーがすべて利用できる必要があるためです。

5 つの地理的リージョンに各クライアントが最寄りの 3 つのクライアントに接続されている場合など、幅広いネットワークのハイブリッドモデルを検討する価値があります。

このシナリオでは、新規クライアントが到達可能なサーバーのサブセットを使用して暗号鍵を設定できます。**tang1**、**tang2** および **tang3** サーバーのセットで、**tang2** が到達できなくなった場合にクライアントは **tang1** および **tang3** で暗号鍵を設定して、後ほど完全なセットを使用して再確立できます。これには、手動による介入、またはより複雑な自動化のいずれかが必要になる場合があります。

17.4.7. Tang サーバーの喪失

同じキー情報が使用されている負荷分散サーバー内の個別の Tang サーバーがなくなると、クライアントからも透過的に確認できます。

同じ URL に関連付けられているすべての Tang サーバー (負荷分散されたセット全体) で一時的に障害が発生すると、ネットワークセグメントの損失と同じと考えることができます。既存クライアントには、事前に設定された Tang サーバーが利用可能な限り、ディスクパーティションを復号化できます。これらのサーバーのいずれかが再びオンラインになるまで、新規クライアントは登録できません。

サーバーを再インストールするか、バックアップからサーバーを復元して、Tang サーバーの物理的な損失を軽減できます。キー情報のバックアップおよび復元プロセスが、権限のないアクセスから適切に保護されていることを確認します。

17.4.8. 危険キー情報の再設定

Tang サーバーの物理的な移動や関連データなど、承認されていないサードパーティーにキーが公開される可能性のある場合は、キーをすぐにローテーションします。

手順

1. 影響を受けた情報を保持する Tang サーバーのキーを再生成します。
2. Tang サーバーを使用してすべてのクライアントのキーを再生成します。
3. 元のキー情報を破棄します。
4. マスター暗号化キーが公開されてしまうという意図しない状況を精査します。可能な場合は、ノードをオフラインにし、ディスクを再暗号化します。

ヒント

同じ物理ハードウェアへの再フォーマットおよび再インストールは時間がかかりますが、自動化およびテストが簡単です。